

Contents

[Azure Architecture Center](#)

[Browse all Architectures](#)

[Architecture icons](#)

[What's new](#)

[Application architecture fundamentals](#)

[Introduction](#)

[Architecture styles](#)

[Overview](#)

[Big compute](#)

[Big data](#)

[Event-driven architecture](#)

[Microservices](#)

[N-tier application](#)

[Web-queue-worker](#)

[Design principles for Azure applications](#)

[Overview](#)

[Design for self-healing](#)

[Make all things redundant](#)

[Minimize coordination](#)

[Design to scale out](#)

[Partition around limits](#)

[Design for operations](#)

[Use managed services](#)

[Use the best data store for the job](#)

[Design for evolution](#)

[Build for the needs of business](#)

[Technology choices](#)

[Choose a compute service](#)

[Azure compute services](#)

- Microservices compute options
- Web apps and single page apps
- Multiparty computing

Choose a container option

- Container options
- Kubernetes at the edge
- Bare-metal Kubernetes at the edge

Choose an identity service

- Active Directory services
- Hybrid identity authentication methods

Choose a storage service

- Storage options
- Managed disk types

Choose a data store

- Understand data store models
- Select a data store
- Criteria for choosing a data store
- Big data storage
- Database scenarios
 - OLAP solutions
 - OLTP solutions
 - Data warehousing
 - Non-relational data store
 - Pipeline orchestration
 - Search data store
 - Data transfer options

Choose an analytics solution

- Analytical data stores
- Analytics and reporting
- Batch processing
- Stream processing

Choose an AI/ML service

- [Cognitive services](#)
 - [Natural language processing](#)
 - [Machine learning](#)
 - [Machine learning tools](#)
 - [Compare MLflow and Azure ML](#)
- [Choose a networking service](#)
 - [Networking options](#)
 - [Load balancing options](#)
 - [VNet peering and VPN gateways](#)
- [Choose a messaging service](#)
 - [Compare messaging services](#)
 - [Asynchronous messaging](#)
 - [Real-time message ingestion](#)
- [Choose an IoT solution](#)
- [Choose a mobile development framework](#)
- [Choose a mixed reality engine](#)
- [Best practices for cloud applications](#)
 - [Overview](#)
 - [API design](#)
 - [API implementation](#)
 - [Autoscaling](#)
 - [Background jobs](#)
 - [Caching](#)
 - [Content Delivery Network](#)
 - [Data partitioning](#)
 - [Data partitioning strategies \(by service\)](#)
 - [Host name preservation](#)
 - [Message encoding considerations](#)
 - [Monitoring and diagnostics](#)
 - [Retry guidance for specific services](#)
 - [Transient fault handling](#)
 - [Performance tuning and antipatterns](#)
 - [Introduction](#)

- [Scenario 1 - Distributed transactions](#)
- [Scenario 2 - Multiple backend services](#)
- [Scenario 3 - Event streaming](#)
- [Performance antipatterns](#)
 - [Overview](#)
 - [Busy Database](#)
 - [Busy Front End](#)
 - [Chatty I/O](#)
 - [Extraneous Fetching](#)
 - [Improper Instantiation](#)
 - [Monolithic Persistence](#)
 - [No Caching](#)
 - [Noisy Neighbor](#)
 - [Retry Storm](#)
 - [Synchronous I/O](#)
- [Responsible engineering](#)
 - [Responsible innovation](#)
 - [Overview](#)
 - [Judgment call](#)
 - [Harms modeling](#)
 - [Understand harm](#)
 - [Assess types of harm](#)
 - [Community jury](#)
 - [Responsible AI](#)
 - [Overview](#)
 - [Six key principles](#)
 - [Guidelines for human-AI interaction](#)
 - [Responsible AI with Cognitive Services](#)
 - [Machine learning](#)
 - [Responsible AI and machine learning](#)
 - [Model interpretability](#)
 - [ML fairness](#)

Differential privacy

Architecture for startups

Overview

Core startup stack architecture

SaaS digital business journey on Azure

Multitenant applications

Overview

Considerations

Overview

Tenancy models

Tenant lifecycle

Pricing models

Measure consumption

Deploy updates

Map requests to tenants

Domain names

Approaches

Overview

Resource organization

Governance and compliance

Cost management and allocation

Deployment and configuration

Compute

Networking

Storage and data

Messaging

AI and ML

Service-specific guidance

Overview

Deployment and configuration

Azure Resource Manager

Compute

- [App Service and Functions](#)
- [Networking](#)
 - [NAT Gateway](#)
- [Storage and data](#)
 - [Azure Storage](#)
 - [Azure SQL Database](#)
 - [Azure Cosmos DB](#)
 - [Azure Database for PostgreSQL](#)
- [Multitenancy checklist](#)
- [Related resources](#)
- [Solutions across Microsoft cloud platforms](#)
- [Azure and Power Platform scenarios](#)
 - [Overview](#)
 - [All Power Platform architectures](#)
 - [CI/CD for Power Platform](#)
 - [Citizen AI with Power Platform](#)
 - [Eventual consistency with Power Apps](#)
 - [Extract text from objects with Power Automate](#)
 - [Power Automate deployment at scale](#)
- [Azure and Microsoft 365 scenarios](#)
 - [Overview](#)
 - [All Microsoft 365 architectures](#)
 - [Governance of Teams guest users](#)
 - [Hybrid SharePoint farm with Microsoft 365](#)
 - [Manage M365 configuration with Azure DevOps](#)
 - [Real-time collaboration](#)
 - [Real-time presence](#)
 - [Secure a Teams channel bot with a firewall](#)
- [Azure and Dynamics 365 scenarios](#)
 - [All Dynamics 365 architectures](#)
 - [Customer 360 with Dynamics 365 CI](#)
 - [Dynamics Business Central as a service](#)

[Enhanced customer dimension with Dynamics 365](#)

[Cloud comparisons](#)

[Azure for AWS professionals](#)

[Overview](#)

[Component information](#)

[Accounts](#)

[Compute](#)

[Databases](#)

[Messaging](#)

[Networking](#)

[Regions and zones](#)

[Resources](#)

[Security and identity](#)

[Storage](#)

[Services comparison](#)

[Azure for Google Cloud professionals](#)

[Overview](#)

[Services comparison](#)

[Design Patterns](#)

[Overview](#)

[Categories](#)

[Data management](#)

[Design and implementation](#)

[Messaging](#)

[Ambassador](#)

[Anti-corruption Layer](#)

[Asynchronous Request-Reply](#)

[Backends for Frontends](#)

[Bulkhead](#)

[Cache-Aside](#)

[Choreography](#)

[Circuit Breaker](#)

Claim Check
Compensating Transaction
Competing Consumers
Compute Resource Consolidation
CQRS
Deployment Stamps
Edge Workload Configuration
Event Sourcing
External Configuration Store
Federated Identity
Gatekeeper
Gateway Aggregation
Gateway Offloading
Gateway Routing
Geode
Health Endpoint Monitoring
Index Table
Leader Election
Materialized View
Pipes and Filters
Priority Queue
Publisher/Subscriber
Queue-Based Load Leveling
Rate Limiting
Retry
Saga
Scheduler Agent Supervisor
Sequential Convoy
Sharding
Sidecar
Static Content Hosting
Strangler Fig

[Throttling](#)

[Valet Key](#)

[Microsoft Azure Well-Architected Framework](#)

[Industry solutions with Azure](#)

[Retail](#)

[Overview](#)

[Guides](#)

[Microsoft Cloud for Retail](#)

[Overview](#)

[Elevate the shopping experience](#)

[Maximize the value of your data](#)

[Security](#)

[Compliance](#)

[Data management in retail](#)

[AI and ML in retail](#)

[Forecast bike rental demand](#)

[Optimize and reuse a recommendation system](#)

[IoT in retail](#)

[Digital distribution center](#)

[In-store analytics checkout](#)

[Create a retail application](#)

[Customize the operator dashboard](#)

[Export data and visualize insights](#)

[Smart inventory management](#)

[Migrate your e-commerce solution to Azure](#)

[SKU optimization for consumer brands](#)

[Visual search in retail with Cosmos DB](#)

[Architectures](#)

[All retail architectures](#)

[AI-based footfall detection](#)

[Build a real-time recommendation API](#)

[Buy online, pickup in store \(retail\)](#)

- [Content-based recommendation](#)
- [E-commerce front end](#)
- [Intelligent search engine for e-commerce](#)
- [Magento e-commerce platform in AKS](#)
- [Movie recommendations on Azure](#)
- [Out of stock detection \(retail\)](#)
- [Scalable order processing](#)
- [Video capture and analytics for retail](#)
- [Compliance solutions](#)
 - [ISO standards](#)
 - [ISO 22301](#)
 - [ISO 27001](#)
 - [ISO 27017](#)
 - [ISO 27018](#)
 - [SOC2 Type 2](#)
 - [PCI DSS](#)
 - [GDPR \(EU\)](#)
- [Finance](#)
 - [Overview](#)
 - [Guides](#)
 - [Microsoft Cloud for Financial Services](#)
 - [Overview](#)
 - [Security](#)
 - [Compliance](#)
 - [Risk grid computing in banking](#)
 - [Risk grid computing solution](#)
 - [Data management in banking](#)
 - [Financial institutions with data mesh](#)
 - [Actuarial risk analysis](#)
 - [Financial services risk lifecycle](#)
 - [Architectures](#)
 - [All finance architectures](#)

- Banking system cloud transformation
- Decentralized trust between banks
- Modernize mainframe & midrange data
- Patterns and implementations in banking
- Real-time fraud detection
- Replicate and sync mainframe data in Azure
- Scale regulated AI and ML in finance
- SWIFT on Azure
 - Alliance Connect
 - SWIFT Alliance Connect
 - SWIFT Alliance Access with Alliance Connect
 - SWIFT AMH with Alliance Connect
 - Alliance Connect Virtual
 - SWIFT Alliance Connect Virtual
 - SWIFT Alliance Access with Alliance Connect Virtual
 - SWIFT AMH with Alliance Connect Virtual
 - SWIFT Alliance Cloud in Azure
 - SWIFT Alliance Lite2
- Compliance solutions
 - PCI DSS overview
 - AKS regulated cluster for PCI
 - AKS regulated - Protect cardholder data
- Healthcare
 - Overview
 - Guides
 - Microsoft Cloud for Healthcare
 - Overview
 - Azure setup
 - Patient engagement
 - Health team collaboration
 - Clinical and operational insights
 - Security in Cloud for Healthcare

Compliance in Cloud for Healthcare

Healthcare APIs

Overview

Healthcare APIs workspace

Text Analytics for health

Overview

Recognized entity categories

Relation extraction

Assertion detection

Architectures

All healthcare architectures

Automate COVID-19 test forms

Build a telehealth system with Azure

Clinical insights with Cloud for Healthcare

Confidential computing for healthcare

Consumer health portal on Azure

Health data consortium

Population health management

Predict hospital readmissions with ML

Precision medicine pipeline

Virtual visits with Cloud for Healthcare

Compliance solutions

EPKS (US) overview

HIPAA and HITRUST

HIPAA (US) overview

HITRUST overview

Implement the healthcare blueprint for AI

HIPAA/HITRUST compliant health data

Government

Overview

Guides

Compare Azure Government and Azure

Considerations for naming resources

Development

Azure Government developer guide

Storage on Azure Government

AI on Azure Government

SSMS on Azure Government

Security

Security for Azure Government

Impact Level 5 isolation

Secure isolation

Secure Azure computing

Compliance

Azure Government compliance

Services compliance scope

Identity

Identity for Azure Government

Integrate Azure AD authentication

Deployment

Deploy with Azure Pipelines

ASE with DISA CAP

Management

Azure Monitor logs

Marketplace

Architectures

All government architectures

Azure Automation in a hybrid environment

Azure Automation update management

Azure Virtual Desktop for the enterprise

Computer forensics chain of custody

Hybrid security monitoring in Azure

Web app private database connectivity

Compliance solutions

[DoD overview](#)

[EAR overview](#)

[FedRAMP](#)

[FedRAMP overview](#)

[FedRAMP high compliance](#)

[FedRAMP moderate compliance](#)

[IRS 1075 overview](#)

[ITAR overview](#)

[NDAA overview](#)

[NIST](#)

[NIST 800-63 overview](#)

[NIST CSF overview](#)

[TIC solutions](#)

[Manufacturing](#)

[Overview](#)

[Guides](#)

[HPC for manufacturing](#)

[Industrial IoT analytics](#)

[Upscale ML lifecycle with MLOps framework](#)

[Predictive maintenance in manufacturing](#)

[Predictive maintenance solution](#)

[Extract actionable insights from IoT data](#)

[Architectures](#)

[All manufacturing architectures](#)

[Anomaly detector process](#)

[Build a speech-to-text transcription pipeline](#)

[Connected factory hierarchy service](#)

[Connected factory signal pipeline](#)

[End-to-end computer vision for manufacturing](#)

[Predictive maintenance with IoT](#)

[Quality assurance](#)

[Supply chain track and trace](#)

Media and Entertainment

[Overview](#)

[Guides](#)

[Dynamics 365 media accelerator](#)

[Content Production solution](#)

[Configure the accelerator with Azure AD](#)

[Architectures](#)

[All media and entertainment architectures](#)

[3D video rendering](#)

[Content-based recommendation](#)

[Digital image-based modeling on Azure](#)

[Gridwich cloud media system](#)

[Image classification on Azure](#)

[Live streaming digital media](#)

[Movie recommendations on Azure](#)

[Compliance solutions](#)

[CDSA overview](#)

[MPA overview](#)

Energy and Environment

[Overview](#)

[Guides](#)

[Microsoft Cloud for Sustainability](#)

[Sustainability outcomes and benefits](#)

[Big compute for oil exploration](#)

[Smart meter energy monitoring](#)

[Solar panel monitoring](#)

[Architectures](#)

[All energy and environment architectures](#)

[Environmental monitoring](#)

[Geospatial data processing and analytics](#)

[Mining equipment monitoring](#)

[Oil and gas tank level forecasting](#)

[Project 15 sustainability](#)

[Run CFD simulations](#)

[Run reservoir simulations](#)

[Compliance solutions](#)

[NERC \(US\) overview](#)

[Game Development](#)

[Overview](#)

[Guides](#)

[Azure PlayFab overview](#)

[Azure AD authentication for PlayFab](#)

[PlayFab CloudScript with Azure Functions](#)

[Multiplayer guidance](#)

[Multiplayer with PlayFab](#)

[Host multiplayer games](#)

[Create virtual machines](#)

[Build definition](#)

[Linux container images](#)

[Measure player latency to Azure](#)

[Connect clients to game servers](#)

[Increase core limits and Azure regions](#)

[PlayFab Insights](#)

[PlayFab samples](#)

[Architectures](#)

[All game development architectures](#)

[AI in games](#)

[Content moderation](#)

[Customer service bot for gaming](#)

[Image classification](#)

[Speech to text for gaming](#)

[Text to speech for gaming](#)

[Text translation for gaming](#)

[Analytics in games](#)

[In-editor debugging telemetry](#)

[Non-real-time dashboard](#)

[Databases for gaming](#)

[Gaming using Azure MySQL](#)

[Gaming using Cosmos DB](#)

[Game streaming](#)

[Unreal Pixel Streaming](#)

[Deploy Unreal Pixel Streaming](#)

[Unreal Pixel Streaming at scale](#)

[Leaderboards](#)

[Leaderboard basics](#)

[Non-relational leaderboard](#)

[Relational leaderboard](#)

[Matchmaking](#)

[Multiplayer matchmaker](#)

[Serverless matchmaker](#)

[Rendering](#)

[3D video rendering](#)

[Digital image-based modeling on Azure](#)

[Scalable gaming servers](#)

[Multiplayer backend architectures](#)

[Real-time multiplayer](#)

[Custom game server scaling](#)

[Multiplayer hosting with Service Fabric](#)

[Multiplayer server hosting with ACI](#)

[Multiplayer server hosting with AKS](#)

[Multiplayer server hosting with Batch](#)

[Turn-based multiplayer](#)

[Asynchronous multiplayer basics](#)

[Serverless asynchronous multiplayer](#)

[Server hosting](#)

[Basic game server hosting](#)

LAMP architectures for gaming

Travel and Hospitality

Overview

Guides

Data science using Spark for airport travel

No-code voice assistant for hospitality

Architectures

All travel and hospitality architectures

Build a chatbot for hotel booking

Build a delta lake in leisure and travel booking

Commerce chatbot as a hotel concierge

Custom business processes for airlines

Migrate a travel web app with APIM

Predictive aircraft engine monitoring

Automotive, Mobility, and Transportation

Overview

Guides

Dynamics 365 automotive accelerator

Azure Maps

Azure Maps traffic coverage

Vehicle consumption model

Single sign on for car park management

Architectures

All automotive architectures

Automated guided vehicles fleet control

Building blocks for autonomous driving

Machine teaching for autonomous vehicles

Predictive insights with vehicle telematics

Process vehicle data using IoT

Real-time asset tracking for vehicles

Run CFD simulations

Compliance solutions

TISAX overview

Telecommunications

- Overview
- Guides
 - Field and cloud edge gateways
 - Edge Workload Configuration pattern
 - Kubernetes at the edge
 - Choose a Kubernetes at the edge option
 - Choose a bare-metal Kubernetes option
 - Dynamics 365 telecommunications accelerator
 - Overview
 - Configure the accelerator for Azure Maps
 - Private multi-access edge compute
 - Overview
 - Partner services
 - Fusion Core
 - Affirmed Private Network Service
 - Azure Network Function Manager
- Architectures
 - All telecommunications architectures
 - Customer churn prediction
 - Deploy AI and ML at the edge
 - Determine customer lifetime and churn
 - Enterprise-grade conversational bot
 - IoT connected light, power, and internet
 - IoT device connectivity for industry
 - Low-latency network connections for industry
 - Predictive maintenance with AI IoT Edge
 - Real-time fraud detection
 - Video capture and analytics
- Compliance solutions
- GSMA overview

Facilities and Real Estate

[Overview](#)

[Guides](#)

[Smart buildings and smart cities ontologies](#)

[Azure Maps indoor maps](#)

[Creator for indoor maps](#)

[Make indoor maps](#)

[Use the Indoor Maps module](#)

[Facility ontology for Azure Maps](#)

[Architectures](#)

[All facilities and real estate architectures](#)

[Cognizant Safe Buildings with IoT](#)

[COVID-19 IoT safe environments](#)

[Facilities management with mixed reality](#)

[IoT connected light, power, and internet](#)

[IoT connectivity for healthcare facilities](#)

[Lighting and disinfection system](#)

[Smart places with Azure Digital Twins](#)

Education

[Overview](#)

[Guides](#)

[Azure Education Hub](#)

[Overview](#)

[Create a lab with REST APIs](#)

[Azure Dev Tools for Teaching](#)

[Dynamics 365 education accelerator](#)

[Identity for education](#)

[Azure Active Directory for education](#)

[Multi-tenant for academic institutions](#)

[Design a tenant configuration](#)

[Design authentication and credentials](#)

[Design an account strategy](#)

Design identity governance

Architectures

All education architectures

Governance of Teams guest users

Moodle deployment with NetApp Files

Secure research for regulated data

Teacher-provisioned virtual labs in Azure

Compliance solutions

FERPA overview

Aerospace

Guides

Azure Orbital for space communication

Modular Datacenter

Drone delivery reference implementation

Domain-driven design for drone delivery

Domain analysis

Tactical DDD

Identify microservice boundaries

Design a microservices drone solution

Introduction

Interservice communication

API design

Data considerations

Monitor drone delivery in production

Performance tuning for drone delivery

Distributed business transactions

Multiple backend services

Event streaming

Architectures

All aerospace architectures

Advanced (AKS) microservices - drones

Predictive maintenance for aircraft monitoring

[Serverless web app for drone delivery](#)
[Vision classifier model - simulated drone](#)

Agriculture

Guides

[Azure FarmBeats overview](#)
[Generate soil moisture heatmap](#)
[Sensor partner integration](#)
[Weather partner integration](#)
[Imagery partner integration](#)

Architectures

[All agriculture architectures](#)
[Environment monitoring with IoT](#)
[Low-latency network for farming](#)

Nonprofit

[Microsoft Cloud for Nonprofit](#)
[Overview](#)
[Set up Microsoft Cloud for Nonprofit](#)
[Donors and supporters](#)
[Fundraising and Engagement](#)
[Fundraising and Engagement overview](#)
[Configure Fundraising and Engagement](#)
[Deploy Fundraising and Engagement](#)
[Dynamics 365 nonprofit accelerator for Azure](#)

Sports

[Analyze Olympic sports with sensors and vision](#)
[Baseball decision analysis with ML.NET](#)

Azure categories

AI + Machine Learning

Get started

Guides

[Cognitive services](#)
[Natural language processing](#)

R developer's guide to Azure

Machine learning

Machine learning options

ML architecture and key concepts

Machine learning at scale

ML pipelines

Compare MLflow and Azure ML

Machine teaching

Enterprise security and governance

Security baseline for AML

MLOps framework

Upscale ML lifecycle with MLOps

MLOps maturity model

Azure ML service selection guide

Industry guidance

Enable the financial services risk lifecycle

Healthcare blueprint for AI

Optimize and reuse recommendations

Predictive maintenance in manufacturing

Predictive maintenance solution

SKU optimization for consumer brands

Visual search for retail

Team Data Science Process

Overview

Lifecycle

Overview

1. Business understanding

2. Data acquisition and understanding

3. Modeling

4. Deployment

5. Customer acceptance

Roles and tasks

Overview

Group manager

Team lead

Project lead

Individual contributor

Project planning

Development

Agile development

Collaborative coding with Git

Execute data science tasks

Code testing

Track progress

Operationalization

DevOps - CI/CD

Training

For data scientists

For DevOps

How To

Set up data science environments

Analyze business needs

Acquire and understand data

Develop models

Deploy models in production

Architectures

AI enrichment with Cognitive Search

Automate document processing

Automate COVID-19 test forms

Baseball decision analysis with ML.NET

Batch scoring for deep learning

Batch scoring with Python

Batch scoring with R

Batch scoring with Spark on Databricks

Build content-based recommendations
Chatbot for hotel booking
Citizen AI with the Power Platform
Deploy AI and ML at the edge
Deploy models to multiple data sources
Determine customer lifetime and churn
Distributed deep learning training
End-to-end computer vision for manufacturing
Enterprise-grade conversational bot
Extract text from objects using Power Automate
Image classification
Implement the healthcare blueprint for AI
Many models ML with Spark
Many models with Azure Machine Learning
MLOps for Python models
Movie recommendations
Orchestrate MLOps on Azure Databricks
Predict hospital readmissions with ML
Quality assurance
Real-time recommendation API
Real-time scoring Python models
Real-time scoring R models
Scale AI and ML in regulated industries
Secure research for regulated data
Speech to text conversion
Training Python models

Solution ideas
AI at the edge
Auditing and risk management
Autonomous systems
Azure Machine Learning architecture
Business process management

Content research
Content tagging with NLP
Contract management
Customer churn prediction
Customer feedback
Data science and machine learning
Defect prevention
Digital asset management
Disconnected AI at the edge
E-commerce chatbot
Energy demand forecasting
Energy supply optimization
Enterprise chatbot disaster recovery
Enterprise productivity chatbot
Environment monitoring
FAQ chatbot
Hospital patient predictions
Image classification with CNNs
Interactive voice response bot
Keyword digital text processing
Marketing optimization
Model deployment to AKS
Personalized marketing solutions
Personalized offers
Population health management
Predictive maintenance
Predictive marketing
Remote patient monitoring
Retail assistant with visual capabilities
Retail product recommendations
Speech services
Vehicle telematics

[Vision classifier model](#)

[Visual assistant](#)

[Analytics](#)

[Get started](#)

[Guides](#)

[Technology choices](#)

[Analytical data stores](#)

[Analytics and reporting](#)

[Batch processing](#)

[Stream processing](#)

[Industry guidance](#)

[Actuarial risk analysis](#)

[Financial institutions with data mesh](#)

[Analytics security baselines](#)

[Security baseline for Azure Data Factory](#)

[Security baseline for Azure Databricks](#)

[Security baseline for Azure Purview](#)

[Apache NiFi guidance](#)

[Apache NiFi monitoring with MonitoFi](#)

[Helm-based deployments for Apache NiFi](#)

[Architectures](#)

[Analytics end to end](#)

[Anomaly detector process](#)

[Apache NiFi on Azure](#)

[Automated enterprise BI](#)

[Big data analytics on confidential computing](#)

[Customer 360 with Synapse and Dynamics 365](#)

[Data analysis for regulated industries](#)

[Data lake queries via Synapse serverless](#)

[Data warehouse for small business](#)

[Data warehousing and analytics](#)

[Employee retention with Databricks and Kubernetes](#)

- Geospatial data processing and analytics
- High throughput stream ingestion
- Ingestion and analysis of news feeds
- Interactive price analytics
- IoT and data analytics
- Long-term security logs in Data Explorer
- Partitioning in Event Hubs and Kafka
- Precision medicine pipeline with genomics
- Stream processing with Azure Databricks
- Stream processing with Stream Analytics
- Stream processing with open-source data

Solution ideas

- Advanced analytics
- App integration using Event Grid
- Big data analytics with Azure Data Explorer
- Big data analytics with enterprise security
- Content Delivery Network analytics
- Data management with Azure Purview
- Demand forecasting for shipping
- Demand forecasting and price optimization
- Demand forecasting with Stream Analytics
- Discovery Hub for analytics
- Enhanced customer dimension
- ETL using HDInsight
- Highly scalable customer service and ERP
- Hybrid big data with HDInsight
- Ingestion, ETL, and stream processing
- Interactive analytics with Data Explorer
- IoT analytics with Azure Data Explorer
- Manage data across the Azure SQL estate
- Mining equipment monitoring
- Modern analytics with Azure Databricks

- Monitoring solution with Data Explorer
- Oil and Gas tank level forecasting
- Predict the length of stay in hospitals
- Predictive aircraft engine monitoring
- Real-time analytics on big data
- Tier applications and data for analytics

Blockchain + Multiparty Compute

- Get started

- Guides

- Azure Confidential Ledger architecture
- Authenticate Confidential Ledger nodes

Architectures

- Azure SQL Database ledger
- Decentralized trust between banks
- Multi-cloud blockchain DLT

Solution ideas

- Blockchain workflow application
- Supply chain track and trace

Compute + HPC

- Get started with HPC

- Guides

- Choose a compute service
- SAS on Azure architecture
- Industry guidance
 - High-performance compute for manufacturing
 - Risk grid computing in banking
 - Risk grid computing solution

VM security baselines

- Security baseline for VM scale sets
- Security baseline for Linux VMs
- Security baseline for Windows VMs

Architectures

- [3D video rendering](#)
- [Computer-aided engineering](#)
- [Digital image modeling](#)
- [HPC cluster deployed in the cloud](#)
- [Linux virtual desktops with Citrix](#)
- [Manage virtual machine compliance](#)
- [Move Azure resources across regions](#)
- [Quantum computing solutions](#)
 - [Loosely coupled quantum computing on Azure](#)
 - [Tightly coupled quantum computing on Azure](#)
- [Run a Linux VM on Azure](#)
- [Run a Windows VM on Azure](#)
- [Run CFD simulations](#)
- [Run reservoir simulations](#)
- [Solution ideas](#)
 - [HPC media rendering](#)
 - [HPC risk analysis](#)
 - [HPC system and big compute](#)
 - [Hybrid HPC with HPC Pack](#)
- [Containers](#)
 - [Get started](#)
 - [Guides](#)
 - [AKS cluster best practices](#)
 - [AKS day-2 operations guide](#)
 - [Introduction](#)
 - [Triage practices](#)
 - [Overview](#)
 - [1- Cluster health](#)
 - [2- Node and pod health](#)
 - [3- Workload deployments](#)
 - [4- Admission controllers](#)
 - [5- Container registry connectivity](#)

- Patching and upgrade guidance
- Monitoring with Azure Monitor
- Common issues
- Choose a Kubernetes option
 - Choose a Kubernetes at the edge option
 - Choose a bare-metal Kubernetes option
- Orchestrate multi-container applications
- Governance disciplines for AKS
- Cost governance with Kubecost
- BCDR considerations for AKS
- Security baseline for AKS
- Docker guidance
 - Monolithic applications
 - Containerize monolithic applications
 - Containers for DevOps collaboration
- Architectures
 - Advanced microservices on AKS
 - AKS baseline cluster
 - AKS baseline for multi-region clusters
 - AKS cluster for a PCI-DSS workload
 - Introduction
 - Architecture
 - Network segmentation
 - Data protection
 - Vulnerability management
 - Access controls
 - Monitoring operations
 - Policy management
 - Summary
 - AKS data protection on Azure NetApp Files
 - Autonomous-driving simulation
 - Build a telehealth system

- Build CNCF projects by using AKS
- CI/CD pipeline for container workloads
- Firewall protection for an AKS cluster
- GitOps for AKS
- Magento e-commerce in AKS
- Microservices architecture on AKS
- Multiplayer server hosting with ACI
- Multiplayer server hosting with AKS
- Multiplayer hosting with Service Fabric
- Multitenancy with AKS and AGIC

Solution ideas

- API-first SaaS business model with AKS
- AKS in event stream processing
- Build cloud native applications
- Bursting from AKS with ACI
- Data streaming with AKS
- Elastic demand handling with AKS
- Instant IoT data streaming with AKS
- JBoss deployment with Red Hat on Azure
- Lift and shift to containers with AKS
- Microservices with AKS and Azure DevOps
- Secure DevOps for AKS

Databases

- Get started
- Guides
- Overview
- Relational data
 - Extract, transform, and load (ETL)
 - Online analytical processing (OLAP)
 - Online transaction processing (OLTP)
- Data warehousing
- Non-relational data

- Non-relational data stores
- Free-form text search
- Time series data
- Working with CSV and JSON files
- Build a scalable system for massive data
- Big data
 - Big data architectures
 - Batch processing
 - Real-time processing
- Technology choices
 - Analytical data stores
 - Analytics and reporting
 - Batch processing
 - Data lakes
 - Data storage
- Choose a data store
 - Understand data store models
 - Select a data store
 - Criteria for choosing a data store
- Pipeline orchestration
- Real-time message ingestion
- Search data stores
- Stream processing
- Data management patterns
- Use the Transactional Outbox pattern
- Industry guidance
 - Data management in banking
 - Data management in retail
 - Financial institutions with data mesh
 - Visual search for retail
- Transfer data to and from Azure
- Extend on-premises data solutions to Azure

Secure data solutions

Tenancy models for SaaS databases

High availability for SQL DB and SQL MI

Cosmos DB guidance

Azure Cosmos DB resource model

Global distribution

Distribute data globally

Consistency levels in Cosmos DB

High availability with Cosmos DB

Global distribution functional details

Partitioning and horizontal scaling

Capacity

Request units in Cosmos DB

Provisioned throughput

Autoscale throughput

Serverless in Cosmos DB

Autoscale and standard throughput

Provisioned and serverless throughput

Scaling provisioned throughput

Security

Security in Cosmos DB

Data encryption

Role-based access control

Azure Policy support

Azure Policy Regulatory Compliance

Security baseline for Cosmos DB

Backup

Backup and restore in Cosmos DB

Continuous backup in Cosmos DB

Cost optimization

Plan and manage costs

Pricing model

Total cost of ownership (TCO)

Understand your bill

Optimize provisioned throughput cost

Optimize request cost

Optimize storage cost

Optimize multi-region cost

Optimize development/testing cost

Optimize cost with reserved capacity

Optimize with rate limiting

Change feed in Cosmos DB

Built-in Jupyter Notebooks support

Service quotas

Monitor Azure Databricks jobs

Overview

Send Databricks application logs

Use dashboards to visualize Databricks

Troubleshoot performance bottlenecks

Run Apache Cassandra

Architectures

Azure health data consortium

Build a delta lake for ad hoc queries

Cost savings through HTAP with Azure SQL

Data governance with Profisee

Data landing zones

Single data landing zone

Multiple data landing zones

Highly sensitive data landing zones

DataOps for modern data warehouse

Globally distributed apps using Cosmos DB

Hybrid ETL with Azure Data Factory

IaaS: Web app with relational database

Master data management with CluedIn

Master data management with Profisee
Migrate master data services with CluedIn
Minimal storage – change feed replication
Multi-region web app with private database
N-tier app with Cassandra
Observability patterns and metrics
Optimize SQL Server with Azure Arc
Optimized storage – time based multi writes
Optimized storage – time based Data Lake
Optimized storage with data classification
Oracle migration to Azure
SQL 2008 R2 failover cluster in Azure
SQL Database and Synapse connectivity
SQL Managed Instance with CMK
Web app private database connectivity
Windows N-tier applications

Solution ideas

Big data analytics with Azure Data Explorer
Build cloud native applications
Campaign optimization with HDInsight
Campaign optimization with SQL Server
Data streaming
Data cache
Digital campaign management
Digital marketing using MySQL
Enterprise data warehouse
Finance management using MySQL
Finance management using PostgreSQL
Gaming using MySQL
Gaming using Cosmos DB
Intelligent apps using MySQL
Intelligent apps using PostgreSQL

- Interactive querying with HDInsight
- Loan charge-off prediction with HDInsight
- Loan charge-off prediction with SQL Server
- Loan credit risk modeling
- Loan credit risk with SQL Server
- Messaging
- Mining equipment monitoring
- Multi-region web app with Cosmos DB
- Ops automation using Event Grid
- Personalization using Cosmos DB
- Retail and e-commerce using MySQL
- Retail and e-commerce using PostgreSQL
- Retail and e-commerce using Cosmos DB
- Serverless apps using Cosmos DB
- Streaming using HDInsight

DataOps

Guides

- DataOps checklist
- Analytics and reporting
- Build a scalable system for massive data
- Apache NiFi guidance
 - Apache NiFi monitoring with MonitoFi
 - Helm-based deployments for Apache NiFi
- Azure Purview guidance
 - Accounts architectures and best practices
 - Managing data effectively
 - Automation best practices
 - Backup and recovery for migration
 - Classification best practices
 - Collections architectures and best practices
 - Deployment best practices
 - Glossary best practices

- [Labeling best practices](#)
- [Data lineage best practices](#)
- [Network architecture and best practices](#)
- [Pricing guidelines](#)
- [Scanning best practices](#)
- [Security best practices](#)

[Architectures](#)

- [Anomaly detector process](#)
- [Apache NiFi on Azure](#)
- [Automated enterprise BI](#)
- [Data analysis for regulated industries](#)
- [Data governance with Profisee](#)
- [Data warehousing and analytics](#)
- [DataOps for modern data warehouse](#)
- [Geospatial data processing and analytics](#)
- [Hybrid ETL with Azure Data Factory](#)
- [Master data management with CluedIn](#)
- [Modern data warehouse for small business](#)

[Solution ideas](#)

- [Azure Data Explorer monitoring](#)
- [Data management with Azure Purview](#)
- [Discovery Hub for analytics](#)
- [Ingestion, ETL, and stream processing](#)
- [Mining equipment monitoring](#)
- [Ops automation using Event Grid](#)
- [Tier applications for analytics](#)

[Developer Options](#)

[Microservices](#)

- [Get started](#)

[Guides](#)

- [Microservices assessment and readiness](#)
- [Domain modeling for microservices](#)

Domain analysis

Tactical DDD

Identify microservice boundaries

Design a microservices architecture

Introduction

Choose a compute option

Interservice communication

API design

API gateways

Data considerations

Container orchestration

Design patterns for microservices

Operate microservices in production

Monitor microservices in AKS

CI/CD for microservices

CI/CD for microservices on Kubernetes

Migrate to a microservices architecture

Migrate monolith to microservices

Modernize apps with Service Fabric

Migrate from Cloud Services to ASF

.NET microservices

Design a microservice-oriented app

Create a CRUD microservice

Event-based communication

Implement API Gateways with Ocelot

Architectures

Decompose apps with Service Fabric

High-availability blue/green deployment

Microservices on Azure Service Fabric

Microservices with Azure Spring Cloud

Unified logging for microservices apps

Serverless applications

Get started

Guides

[Serverless Functions examples](#)

[Plan for serverless architecture](#)

[Deploy serverless Functions](#)

[Serverless application assessment](#)

[Technical workshops and training](#)

[Proof of concept or pilot](#)

[Develop and deploy serverless apps](#)

[Serverless Functions app development](#)

[Serverless Functions code walkthrough](#)

[CI/CD for a serverless frontend](#)

[Monitoring serverless event processing](#)

[Serverless Functions app operations](#)

[Serverless Functions app security](#)

[Security baseline for Azure Functions](#)

[Serverless with Azure Logic Apps](#)

[Event Hubs with Azure Functions](#)

[Overview](#)

[Performance and scale](#)

[Resilient design](#)

[Security](#)

[Observability](#)

Architectures

[Azure Functions in a hybrid environment](#)

[Event-based cloud automation](#)

[Multicloud with the Serverless Framework](#)

[Real-time location sharing](#)

[Serverless event processing](#)

Solution ideas

[AKS in event stream processing](#)

[Big data analytics with Data Explorer](#)

- De-batch and filter with Event Hubs
- Serverless applications using Event Grid
- Serverless apps using Cosmos DB
- Serverless computing LOB apps
- Serverless event stream processing
- Transit Hub pub-sub messaging system

DevOps

- Get started

- Guides

- DevOps checklist

- Operational Excellence patterns

- Advanced ARM templates

- Overview

- Update a resource

- Use an object as a parameter

- Property transformer and collector

- DevTest Labs guidance

- Deliver a proof of concept

- Orchestrate the implementation

- Scale up DevTest Labs infrastructure

- Security baseline for DevTest Labs

- Azure Monitor guidance

- Best practices

- Plan your monitoring strategy

- Configure data collection

- Analyze and visualize data

- Alerts and automated actions

- Continuous monitoring

- Data sources

- Data platform

- Security

- Log data security

[Customer-managed keys](#)

[Private Link connection](#)

[Private Link design](#)

[Personal log data handling](#)

[Data collection, retention, and storage](#)

[Security baseline](#)

[Design a logging deployment](#)

[CI/CD for Synapse Analytics](#)

[DevOps for quantum computing](#)

[Platform automation for VMware Solution](#)

[Architectures](#)

[Automate multistage DevOps pipelines](#)

[Automate Sentinel integration with Azure DevOps](#)

[Automated API deployments using APIOps](#)

[Automated Jupyter Notebooks for diagnostics](#)

[Azure DevTest Labs for enterprises](#)

[CI/CD pipeline for chatbots with ARM](#)

[CI/CD pipeline using Azure DevOps](#)

[DevSecOps in GitHub](#)

[Enterprise monitoring with Azure Monitor](#)

[High-availability blue/green deployment](#)

[Jenkins on Azure](#)

[Microsoft 365 tenant configuration](#)

[Run containers in a hybrid environment](#)

[Teacher-provisioned virtual labs in Azure](#)

[Solution ideas](#)

[CI/CD for Azure VMs](#)

[CI/CD for Azure Web Apps](#)

[CI/CD for Containers](#)

[CI/CD for Microsoft Power Platform](#)

[CI/CD for quantum computing jobs](#)

[CI/CD for Windows desktop apps](#)

- [CI/CD using Jenkins and AKS](#)
- [CI/CD using Jenkins and Terraform](#)
- [DevSecOps in Azure](#)
- [DevSecOps with a rolling main branch](#)
- [DevTest and DevOps for IaaS](#)
- [DevTest and DevOps for PaaS](#)
- [DevTest and DevOps with microservices](#)
- [DevTest Image Factory](#)
- [Hybrid DevOps](#)
- [Java CI/CD using Jenkins and Web Apps](#)
- [SharePoint for development testing](#)
- [Hybrid + Multicloud](#)
 - [Get started](#)
 - [Guides](#)
 - [Hybrid workload in Azure](#)
 - [Hybrid app design considerations](#)
 - [Azure Arc guidance](#)
 - [Administer SQL Server with Azure Arc](#)
 - [App Service on Azure Arc](#)
 - [Security baseline for Arc-enabled Servers](#)
 - [Hybrid deployments](#)
 - [Configure hybrid cloud connectivity](#)
 - [Configure hybrid cloud identity](#)
 - [Deploy AI-based footfall detection](#)
 - [Deploy an app that scales cross-cloud](#)
 - [Deploy Kubernetes on Azure Stack Hub](#)
 - [Deploy highly available MongoDB](#)
 - [Deploy hybrid app with on-premises data](#)
 - [Deploy a SQL Server 2016 AG](#)
 - [Direct traffic with a geo-distributed app](#)
 - [Azure VMware Solution guidance](#)
 - [API Management for AVS](#)

[BCDR for AVS](#)

[Govern AVS](#)

[Manage and monitor AVS](#)

[Migrate with AVS](#)

[Network interconnectivity for AVS](#)

[Network planning for AVS](#)

[Platform automation for AVS](#)

[Security and governance for AVS](#)

[Connect an on-premises network to Azure](#)

[FSLogix for the enterprise](#)

[Troubleshoot a hybrid VPN connection](#)

[Architectures](#)

[Azure Arc solutions](#)

[Azure Arc hybrid management with AKS](#)

[Manage configurations for Azure Arc](#)

[Optimize SQL Server with Azure Arc](#)

[Run containers in a hybrid environment](#)

[Azure Automation solutions](#)

[Azure Automation hybrid environment](#)

[Azure Automation update management](#)

[Azure Automation State Configuration](#)

[Azure enterprise cloud file share](#)

[Azure files secured by AD DS](#)

[Azure Functions in a hybrid environment](#)

[Azure Stack HCI stretched clusters for DR](#)

[Azure Stack HCI switchless interconnect](#)

[Azure Stack Hub solutions](#)

[AI-based footfall detection](#)

[Back up files on Azure Stack Hub](#)

[Cross-cloud scaling \(on-premises data\)](#)

[Cross-cloud scaling with Traffic Manager](#)

[DevOps with Azure Stack Hub](#)

- [Disaster recovery for Stack Hub VMs](#)
- [Hybrid geo-distributed architecture](#)
- [Hybrid relay connection](#)
- [Azure VMware Solution in hub-and-spoke](#)
- [Connect on-premises with ExpressRoute](#)
- [Connect standalone servers](#)
- [Deploy AI and ML with Azure Stack Edge](#)
- [Design a hybrid Domain Name System](#)
- [Enhanced-security hybrid messaging: client](#)
- [Enhanced-security hybrid messaging: mobile](#)
- [Enhanced-security hybrid messaging: web](#)
- [Extend on-premises using ExpressRoute](#)
- [Extend an on-premises network using VPN](#)
- [Hybrid availability and monitoring](#)
- [Hybrid file services](#)
- [Hybrid file share with disaster recovery](#)
- [Hybrid security monitoring](#)
- [Manage hybrid workloads with WAC](#)
- [On-premises data gateway for Logic Apps](#)
- [Public MEC compute deployment](#)
- [Public MEC high availability](#)
- [Run containers in a hybrid environment](#)
- [Use Azure file shares in a hybrid environment](#)
- [Solution ideas](#)
 - [Azure Stack Hub solutions](#)
 - [Cross-cloud scaling](#)
 - [Hybrid connections](#)
 - [Hybrid relay connection](#)
 - [Tiered data for analytics](#)
 - [Unlock legacy data with Azure Stack](#)
 - [Azure VMware Solution foundations](#)
 - [Azure VMware Solution capacity planning](#)

[Azure VMware Solution landing zone](#)

[Azure VMware Solution networking](#)

[Cross-platform chat](#)

[Identity](#)

[Get started](#)

[Guides](#)

[Azure Active Directory architecture](#)

[Identity and access management in Azure](#)

[Compare identity services](#)

[Build for resilience](#)

[Conditional Access](#)

[Conditional Access for Zero trust](#)

[Conditional Access design principles](#)

[Conditional Access architecture](#)

[Conditional Access framework and policies](#)

[Identity in multitenant applications](#)

[Introduction](#)

[The Tailspin scenario](#)

[Authentication](#)

[Claims-based identity](#)

[Tenant sign-up](#)

[Application roles](#)

[Authorization](#)

[Secure a web API](#)

[Cache access tokens](#)

[Client certificate](#)

[Federate with a customer's AD FS](#)

[Integrate on-premises AD with Azure](#)

[Deployment guidance](#)

[Azure AD deployment checklist](#)

[Azure AD deployment plans](#)

[Azure AD B2C deployment plans](#)

[Migrate applications to Azure AD](#)

[Migrate an AD FS app to Azure](#)

[Migrate app authentication to Azure AD](#)

[Use the AD FS application activity report](#)

[Resources for migrating to Azure AD](#)

[Secure identity](#)

[Secure development with SPAs](#)

[Security baseline for Azure AD](#)

[Identity management](#)

[Azure billing and AAD tenants](#)

[Identity and access management](#)

[Limit cross-tenant private endpoints](#)

[Resource organization](#)

[Hybrid identity](#)

[Choose a hybrid identity method](#)

[Cloud management for on-premises](#)

[Hybrid identity foundation with Azure AD](#)

[Azure AD Connect for on-premises](#)

[Identity for education](#)

[Azure Active Directory for education](#)

[Multi-tenant for large institutions](#)

[Design a tenant configuration](#)

[Design authentication and credentials](#)

[Design an account strategy](#)

[Design identity governance](#)

[Architectures](#)

[AD DS resource forests in Azure](#)

[Azure AD identity management for AWS](#)

[Deploy AD DS in an Azure virtual network](#)

[Extend on-premises AD FS to Azure](#)

[Governance of Teams guest users](#)

[Hybrid identity](#)

On-premises AD domains with Azure AD
Solution ideas

Collaboration with Microsoft 365
Integration

Guides
Connectors in Azure Logic Apps
Access to virtual networks from Logic Apps
BCDR for Logic Apps
Azure Policy controls for Logic Apps
Security baseline for Logic Apps

Architectures
Basic enterprise integration on Azure
Data integration with Logic Apps and SQL
Enterprise business intelligence
Enterprise integration - queues and events
On-premises data gateway for Logic Apps
Power Automate deployment at scale
Publish internal APIs to external users

Solution ideas
Custom business processes
Elastic Workplace Search on Azure
Line of business extension
Web and mobile front-ends

Internet of Things
Get started
Guides
IoT concepts
Introduction to IoT solutions
Devices, platform, and applications
Attestation, authentication, provisioning
Field and cloud edge gateways
Application-to-device commands

Builders, developers, and operators

Computer vision with Azure IoT Edge

- Overview
- Camera selection
- Hardware acceleration
- Machine learning
- Alerting
- Image storage
- User interface and scenarios

Industrial IoT analytics

- Architecture
- Recommended services
- Data visualization
- Considerations

IoT patterns

- Analyze and optimize loop
- Event routing
- Measure and control loop
- Monitor and manage loop
- Real-time IoT updates
- Scale solutions with deployment stamps

Azure IoT client SDK support

Choose an IoT solution

Moving from test to production

Architectures

- Azure IoT reference architecture
- Automated guided vehicles fleet control
- Computer vision at the edge
- Connected factory hierarchy service
- Connected factory signal pipeline
- Efficient Docker image deployment
- IoT and data analytics

[IoT using Cosmos DB](#)

[Predictive maintenance with IoT](#)

[Smart places with Azure Digital Twins](#)

[Solution ideas](#)

[Azure digital twins builder](#)

[Buy online, pick up in store](#)

[Condition monitoring](#)

[COVID-19 solutions](#)

[Cognizant Safe Buildings with IoT](#)

[Contactless IoT interfaces](#)

[COVID-19 IoT safe environments](#)

[IoT Connected Platform](#)

[Lighting and disinfection system](#)

[Environment monitoring](#)

[IoT analytics with Azure Data Explorer](#)

[IoT Edge data storage and processing](#)

[Light and power for emerging markets](#)

[Process real-time vehicle data using IoT](#)

[Project 15 IoT sustainability](#)

[Real-time asset tracking and management](#)

[Voice assistants and IoT devices](#)

[Mainframe + Midrange](#)

[Get started](#)

[Guides](#)

[Mainframe migration framework](#)

[Mainframe migration overview](#)

[Mainframe myths and facts](#)

[Mainframe migration strategies](#)

[Mainframe application migration](#)

[Mainframe rehosting](#)

[Mainframe rehosting on Azure VMs](#)

[Move mainframe compute to Azure](#)

[Move mainframe storage to Azure](#)

[Get started with TmaxSoft OpenFrame](#)

[App modernization](#)

[Architectures](#)

[AIX UNIX to Azure Linux migration](#)

[Batch transaction processing](#)

[General mainframe refactor to Azure](#)

[IBM System i to Azure using Infinite i](#)

[IBM z/OS migration with Asysco AMT](#)

[IBM z/OS online transaction processing](#)

[Integrate IBM MQs with Azure](#)

[Micro Focus Enterprise Server on Azure](#)

[Migrate AIX workloads with Skytap](#)

[Migrate IBM i series to Azure with Skytap](#)

[Refactor IBM z/OS mainframe CF](#)

[Refactor mainframe apps with Advanced](#)

[Refactor systems that run Adabas & Natural](#)

[Refactor mainframe with Raincode](#)

[Unisys CPF rehost using virtualization](#)

[Unisys Dorado migration](#)

[Unisys mainframe migration with Asysco](#)

[Use LzLabs SDM in Azure](#)

[Solution ideas](#)

[Migrate IBM apps with TmaxSoft](#)

[Solaris emulator on Azure VMs](#)

[Data modernization](#)

[Architectures](#)

[Mainframe data replication with Qlik](#)

[Modernize mainframe and midrange data](#)

[Re-engineer mainframe batch apps](#)

[Replicate and sync mainframe data](#)

[Solution ideas](#)

Mainframe access to Azure databases

Mainframe file replication on Azure

Management + Governance

Guides

Governance best practices

Backup

Backup architecture and components

Azure Backup support matrix

Backup cloud and on-premises workloads

Disaster recovery

Azure to Azure disaster recovery

Support matrix for Azure VM DR

ExpressRoute with Azure VM DR

Move Azure VMs to another Azure region

BCDR for Azure VMware Solution

Management for Azure environments

Management for VMware Solution

Architectures

Back up cloud applications

Computer forensics

End-to-end governance when using CI/CD

Highly available SharePoint Server 2016

Hybrid management

Azure Arc hybrid management with AKS

Azure Automation hybrid environment

Azure Automation Update Management

Back up files on Azure Stack Hub

Disaster recovery for Azure Stack Hub

Hybrid availability and monitoring

Manage configurations for Azure Arc

Manage hybrid workloads with WAC

Updating Windows VMs in Azure

Solution ideas

- Archive on-premises data to cloud
- Back up on-premises applications
- Centralize app configuration and security
- Data Sovereignty & Data Gravity
- Enterprise-scale disaster recovery
- High availability for BCDR
- SMB disaster recovery with Site Recovery
- SMB disaster recovery with Double-Take DR

Media

Get started

Guides

- Media Services terminology and concepts
- Encoding video and audio
- Live streaming
- High availability with video on demand
- Monitor Media Services
- Dynamic encryption
- Security baseline for Media Services

Architectures

Gridwich media processing system

Gridwich architecture

Gridwich concepts

Clean monolith design

Saga orchestration

Project names and structure

Gridwich CI/CD

Content protection and DRM

Gridwich Media Services

Gridwich Storage Service

Gridwich logging

Gridwich message formats

Pipeline variables to Terraform flow

Gridwich procedures

Set up Azure DevOps

Run Azure admin scripts

Set up local dev environment

Create new cloud environment

Maintain and rotate keys

Test Media Services V3 encoding

Solution ideas

Instant broadcasting with serverless

Live streaming digital media

Video-on-demand digital media

Migration

Guides

Migrate an e-commerce solution to Azure

Migrate with Azure VMware Solution

Architectures

Banking system

Banking cloud transformation

Patterns and implementations

JMeter implementation reference

Lift and shift LOB apps

Oracle database migration

Migration decision process

Cross-cloud connectivity

Lift and shift to Azure VMs

Refactor

Rearchitect

Mixed Reality

Guides

Azure mixed reality cloud services

Choose a mixed reality engine

Design and prototype for mixed reality

Spatial Anchors in a shared experience

MR core concepts

App model

App views

Audio in mixed reality

Comfort

Coordinate systems

Eye tracking

Holographic frame

Room scan visualization

Scene understanding

Spatial anchors

Spatial mapping

Start gesture

Types of mixed reality apps

Design content for holographic display

Shared experiences in mixed reality

Interaction models

Instinctual interactions

Hands and motion controllers

Direct manipulation with hands

Motion controllers

Point and commit with hands

Hands-free model

Hands-free

Voice input

Eye-gaze and dwell

Head-gaze and dwell

Eye-gaze-based interaction

Gaze and commit model

Gaze and commit

Head-gaze and commit

UI controls and behaviors

Cursors

Point and commit with hands

Button

Interactable object

Bounding box and App bar

Direct manipulation with hands

Hand menu

Near menu

Object collection

Keyboard

Tooltip

Slate

Slider

Shader

Dialog

Hand coach

Spatial mesh

Billboarding and tag-along

Progress indicator

Surface magnetism

Solution ideas

Design review with mixed reality

Facilities management with mixed reality

Training powered by mixed reality

Mobile

Guides

Choose a mobile development framework

Build a serverless mobile back-end

Cloud-hosted source control for mobile

Continuous build and integration for mobile

- Continuous delivery for mobile apps
- Add authentication in mobile apps
- Store, sync, and query mobile app data
- Cloud storage for mobile apps
- Analyze mobile app use

Solution ideas

- Adding mobile front-ends to legacy apps
- Custom mobile workforce app
- Scalable apps with Azure MySQL
- Scalable apps using Azure PostgreSQL
- Social app for with authentication
- Task-based consumer mobile app

Networking

Get started

Guides

Network security

- Azure network security overview
- Best practices for network security
- Application Gateway framework review
- Azure Firewall guidance
 - Azure Firewall architecture overview
 - Azure Firewall framework review
 - Security baseline for Azure Firewall

NAT gateway framework review

Private Link in hub-and-spoke network

Virtual Network guidance

- Add IP spaces to peered virtual networks
- Segment virtual networks
- VNet peering and VPN gateways
- Limit cross-tenant private endpoints
- Build solutions with availability zones
- Use ExpressRoute with Power Platform

Overview

Benefits of using ExpressRoute

How ExpressRoute works

Before you use ExpressRoute

Understand Power Platform architecture

Plan an ExpressRoute deployment

Set up ExpressRoute for Power Platform

ExpressRoute readiness checklist

ExpressRoute for Office 365

Connectivity to other cloud providers

Connectivity to Oracle Cloud Infrastructure

Architectures

Deploy highly available NVAs

Enterprise-scale landing zone

Global transit network and Virtual WAN

High availability for IaaS apps

Hub-spoke network topology in Azure

Hub-spoke topology with Virtual WAN

Hybrid networking

Azure Automation Update Management

Connect servers with Network Adapter

Design a hybrid DNS solution

Hybrid availability and monitoring

Implement a secure hybrid network

Interconnect with China using Virtual WAN

Migrate to Azure Virtual WAN

Multi-region N-tier application

Multi-region load balancing

Multi-tier web application built for HA/DR

Multitenant SaaS

Network-hardened web app

Network topology and connectivity with AVS

- Private Link and DNS integration at scale
- SD-WAN connectivity with Virtual WAN
- Traditional Azure networking topology
- Trusted Internet Connections (TIC) 3.0 compliance
- Update route tables with Route Server
- Virtual WAN network topology
- Virtual WAN optimized for requirements

Solution ideas

- Low-latency network for industry
- Video capture and analytics for retail
- IoT network for healthcare facilities

Oracle

Guides

- Connectivity to Oracle Cloud Infrastructure
- Oracle solutions on Azure
- Design an Oracle database in Azure
- Oracle Database backup
 - Oracle Database backup strategies
 - Oracle backup using Azure Storage
 - Oracle backup using Azure Backup
- Oracle disaster recovery options
- Oracle WebLogic Server
 - Oracle WebLogic Server on Azure VMs
 - Oracle WebLogic Server on AKS

Architectures

- Oracle application architectures
- Oracle Cloud Infrastructure solutions
- Oracle database architectures
- Oracle database migration
 - Oracle database migration to Azure
- Migration decision process
- Cross-cloud connectivity

[Lift and shift to Azure VMs](#)

[Refactor](#)

[Rearchitect](#)

[Oracle Database with Azure NetApp Files](#)

[Run Oracle databases on Azure](#)

[SAP deployment using an Oracle database](#)

[SAP system on Oracle Database](#)

[SAP](#)

[Get started](#)

[Guides](#)

[SAP checklist](#)

[SAP HANA infrastructure configurations](#)

[SAP workload configurations with AZs](#)

[Supported scenarios for HLI](#)

[ANF volume group for SAP HANA](#)

[Architectures](#)

[Dev/test for SAP](#)

[SAP BusinessObjects BI platform](#)

[SAP BusinessObjects BI platform for Linux](#)

[SAP BW/4HANA in Linux on Azure](#)

[SAP deployment using an Oracle DB](#)

[SAP HANA on HLI general architecture](#)

[SAP HANA on HLI network architecture](#)

[SAP HANA on HLI with HA and DR](#)

[SAP HANA scale-out with standby node](#)

[SAP HANA scale-up on Linux](#)

[SAP NetWeaver on Windows on Azure](#)

[SAP S/4HANA in Linux on Azure](#)

[SAP system on Oracle Database on Azure](#)

[Solution ideas](#)

[SAP NetWeaver on SQL Server](#)

[SAP S/4 HANA for Large Instances](#)

SAP workload automation using SUSE

Security

[Get started](#)

Guides

[Security design principles](#)

[SecOps best practices](#)

[Highly-secure IaaS apps](#)

[Microsoft Cybersecurity Reference Architectures](#)

[Virtual Network security options](#)

[Zero-trust network for web applications](#)

[Azure governance design area](#)

[Integrate Data Explorer with Sentinel](#)

Architectures

[Automate Sentinel integration with Azure DevOps](#)

[Azure AD in Security Operations](#)

[Cyber threat intelligence](#)

[Healthcare platform confidential computing](#)

[Homomorphic encryption with SEAL](#)

[Hybrid security monitoring](#)

[Improved-security access to multitenant](#)

[Long-term security logs in Data Explorer](#)

[MCAS and Azure Sentinel security for AWS](#)

[Multilayered protection for Azure VMs](#)

[Real-time fraud detection](#)

[Restrict interservice communications](#)

[Secure OBO refresh tokens](#)

[Securely managed web apps](#)

[Secure a Microsoft Teams channel bot](#)

[Secure research for regulated data](#)

[SQL Managed Instance with CMK](#)

[Virtual network integrated microservices](#)

[Web app private database connectivity](#)

Storage

[Get started](#)

[Guides](#)

[Storage accounts](#)

[Security and compliance](#)

[Storage encryption for data at rest](#)

[Azure Policy controls for Storage](#)

[Use private endpoints](#)

[Security baseline for Azure Storage](#)

[Data redundancy](#)

[DR and storage account failover](#)

[Azure Storage migration guidance](#)

[FSLogix for the enterprise](#)

[Blob storage guidance](#)

[Authorize access to blobs with AAD](#)

[Authorize access with role conditions](#)

[Data protection](#)

[Security recommendations](#)

[Performance and scalability checklist](#)

[Data Lake Storage guidance](#)

[Best practices](#)

[Security controls by Azure Policy](#)

[File storage guidance](#)

[Planning for deployment](#)

[Identity-based authentication](#)

[Networking considerations](#)

[Disaster recovery and failover](#)

[File share backup](#)

[Queue storage guidance](#)

[Authorize access with AAD](#)

[Performance and scalability checklist](#)

[Table storage guidance](#)

- [Authorize access with AAD](#)
- [Performance and scalability checklist](#)
- [Design scalable and performant tables](#)
- [Design for querying](#)
- [Table design patterns](#)
- [Disk storage guidance](#)
 - [Choose a managed disk type](#)
 - [Server-side encryption](#)
 - [Disk Encryption for Linux VMs](#)
 - [Disk Encryption for Windows VMs](#)
 - [Design for high performance](#)
 - [Scalability and performance targets](#)
 - [Create an incremental snapshot](#)
- [Azure NetApp Files guidance](#)
 - [Solution architectures](#)
 - [Use ANF with Oracle Database](#)
 - [ANF for electronic design automation](#)
 - [Use ANF with Virtual Desktop](#)
 - [Use ANF with SQL Server](#)
 - [ANF application volume group for SAP HANA](#)
- [Architectures](#)
 - [Azure file shares in a hybrid environment](#)
 - [Azure files secured by AD DS](#)
 - [Azure NetApp Files solutions](#)
 - [AKS data protection on ANF](#)
 - [Enterprise file shares with DR](#)
 - [Moodle deployment with ANF](#)
 - [Oracle Database with Azure NetApp Files](#)
 - [SQL Server on VMs with ANF](#)
 - [Hybrid file services](#)
 - [Minimal storage – change feed replication](#)
 - [Multi-region web app with replication](#)

- Optimized storage data classification
- Solution ideas
 - HIPAA/HITRUST Health Data and AI
 - Media rendering
 - Medical data storage
 - Two-region app with Table storage failover
- Virtual Desktop
 - Guides
 - Authentication in Azure Virtual Desktop
 - Azure Virtual Desktop network connectivity
 - Azure AD join for Azure Virtual Desktop
 - Connect RDP Shortpath
 - FSLogix guidance
 - FSLogix for the enterprise
 - FSLogix profile containers and files
 - Storage FSLogix profile container
- Architectures
 - Azure Virtual Desktop for the enterprise
 - Multiple Active Directory forests
 - Multiple forests with Azure AD DS
- Web
 - Get started
 - Guides
 - Design principles
 - Design and implementation patterns
 - App Service considerations
 - Deployment best practices
 - Security recommendations
 - Security baseline for App Service
 - Networking features for App Service
 - Modernize web apps
 - Characteristics of modern web apps

- [Choose between web apps and SPAs](#)
- [ASP.NET architectural principles](#)
- [Common client-side web technologies](#)
- [Development process for Azure](#)
- [Azure hosting for ASP.NET web apps](#)
- [Eventual consistency in Power Platform](#)
- [Architectures](#)
 - [Basic web application](#)
 - [Clinical insights with Cloud for Healthcare](#)
 - [Common web app architectures](#)
 - [Consumer health portal](#)
 - [Deployment in App Service Environments](#)
 - [Standard deployment](#)
 - [High availability deployment](#)
 - [E-commerce front end](#)
 - [Highly available multi-region web app](#)
 - [IaaS: Web app with relational database](#)
 - [Improved-security access to multitenant web apps from on-premises](#)
 - [Intelligent search engine for e-commerce](#)
 - [Migrate a web app using Azure APIM](#)
 - [Multi-region web app with private database](#)
 - [Multi-tier app service with private endpoint](#)
 - [Multi-tier app service with service endpoint](#)
 - [Multi-tier web app built for HA/DR](#)
 - [Protect APIs with Application Gateway](#)
 - [Real-time location sharing](#)
 - [Scalable and secure WordPress on Azure](#)
 - [Scalable cloud applications and SRE](#)
 - [Scalable order processing](#)
 - [Scalable web app](#)
 - [Serverless web app](#)
 - [Virtual visits with Cloud for Healthcare](#)

Web app monitoring on Azure

Web app private database connectivity

Solution ideas

Dynamics Business Central as a service

E-commerce website running in ASE

Highly available SharePoint farm

Hybrid SharePoint farm with Microsoft 365

Real-time presence with Microsoft 365

Scalable e-commerce web app

Scalable Episerver marketing website

Scalable Sitecore marketing website

Scalable Umbraco CMS web app

Scalable web apps with Redis

Simple branded website

Simple digital marketing website

Web/mobile apps with MySQL and Redis

Cloud Adoption Framework

Azure architecture icons

3/10/2022 • 2 minutes to read • [Edit Online](#)

Helping our customers design and architect new solutions is core to the Azure Architecture Center's mission. Architecture diagrams like those included in our guidance can help communicate design decisions and the relationships between components of a given workload. On this page you will find an official collection of Azure architecture icons including Azure product icons to help you build a custom architecture diagram for your next solution.

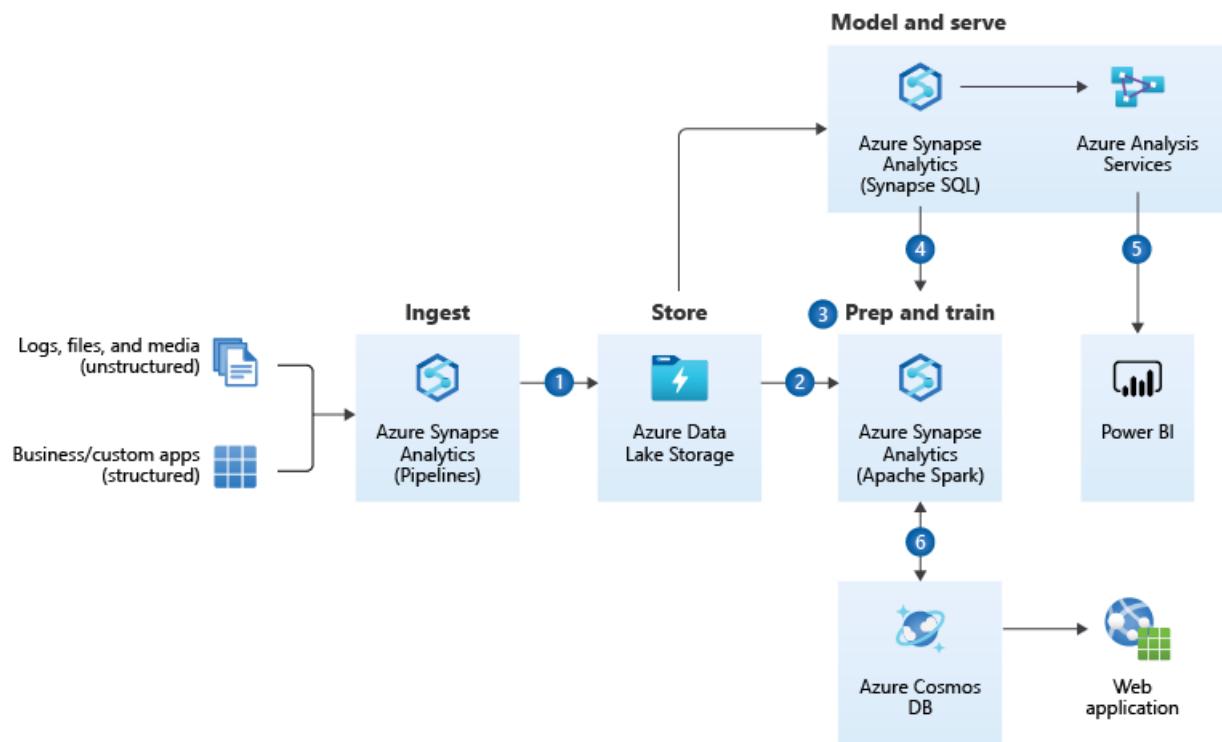
Do's

- Use the icon to illustrate how products can work together
- In diagrams, we recommend to include the product name somewhere close to the icon
- Use the icons as they would appear within Azure

Don'ts

- Don't crop, flip or rotate icons
- Don't distort or change icon shape in any way
- Don't use Microsoft product icons to represent your product or service

Example architecture diagram



[Browse all Azure architectures](#) to view other examples.

Icon updates

November 2020

The folder structure of our collection of Azure architecture icons has changed. The FAQs and Terms of Use PDF files appear in the first level when you download the SVG icons below. The files in the icons folder are the same except there is no longer a CXP folder. If you encounter any issues, let us know.

January 2021

There are ~26 icons that have been added to the existing set. The download file name has been updated to Azure_Public_Service_Icons_V4.zip

Icon terms

Microsoft permits the use of these icons in architectural diagrams, training materials, or documentation. You may copy, distribute, and display the icons only for the permitted use unless granted explicit permission by Microsoft. Microsoft reserves all other rights.

I agree to the above terms

DOWNLOAD SVG
ICONS

More icon sets from Microsoft

- [Microsoft 365 architecture icons and templates](#)
- [Dynamics 365 icons](#)
- [Microsoft Power Platform icons](#)

What's new in Azure Architecture Center

3/10/2022 • 6 minutes to read • [Edit Online](#)

The Azure Architecture Center (AAC) helps you design, build, and operate solutions on Azure. Learn about the cloud architectural styles and design patterns. Use the technology choices and guides to decide the services that are right for your solution. The guidance is based on all aspects of building for the cloud, such as operations, security, reliability, performance, and cost optimization.

New and updated articles in Azure Architecture Center.

March 2022

New Articles

- [DevOps architecture design](#)
- [Automation of COVID-19 test forms](#)
- [SWIFT Alliance Cloud in Azure](#)
- [SWIFT's Alliance Access in Azure](#)
- [SWIFT's Alliance Access with Alliance Connect Virtual in Azure](#)
- [SWIFT's Alliance Messaging Hub \(AMH\) with Alliance Connect](#)
- [SWIFT's Alliance Messaging Hub \(AMH\) with Alliance Connect Virtual](#)
- [SWIFT Alliance Lite2 on Azure](#)
- [SWIFT Alliance Connect Virtual in Azure](#)
- [Automate Sentinel integration with Azure DevOps](#)
- [Refactor mainframe computer systems that run Adabas & Natural](#)
- [Host name preservation](#)
- [Employee retention with Databricks and Kubernetes](#)

Updated Articles

- [Tenancy model for SaaS applications \(#e2c5be9eb\)](#)
- [Build a chatbot for hotel booking \(#abfbdb23b\)](#)
- [Citizen AI with the Power Platform \(#7636a039e\)](#)
- [SAP S/4HANA in Linux on Azure \(#f032d0008\)](#)

February 2022

New Articles

- [SWIFT's Alliance Messaging Hub \(AMH\) with Alliance Connect Virtual](#)
- [Web architecture design](#)
- [SWIFT's Alliance Access with Alliance Connect Virtual in Azure](#)
- [Azure public multi-access edge compute deployment](#)
- [Multi-tier app service with private endpoint](#)
- [Azure and Microsoft 365 scenarios](#)
- [Big data analytics on confidential computing](#)
- [Multitenancy checklist on Azure](#)

- Solutions for the telecommunications industry
- Real-time collaboration with Azure and Microsoft 365
- Real-time presence with Microsoft 365, Azure, and Power Platform
- Networking architecture design
- Azure and Power Platform scenarios
- Architectural approaches for AI and ML in multitenant solutions
- Tiered data for analytics
- Architectural approaches for messaging in multitenant solutions
- Trusted Internet Connection (TIC) 3.0 compliance
- Actuarial risk analysis and financial modeling
- Data management in banking
- Enable the financial services risk lifecycle with Azure and R
- Risk grid computing in banking
- Risk grid computing solution
- Implement the Azure healthcare blueprint for AI
- High-performance computing (HPC) for manufacturing
- Extract actionable insights from IoT data
- Introduction to predictive maintenance in manufacturing
- Predictive maintenance solution
- Migrate your e-commerce solution to Azure
- Reuse recommender systems and algorithms from R with Azure
- Data management in the retail industry
- SKU optimization for consumer brands
- Visual search in retail with Cosmos DB
- Solutions for the education industry
- Security architecture design
- Azure Database for PostgreSQL considerations for multitenancy
- Hybrid geo-distributed architecture
- Cross-cloud scaling with Traffic Manager
- Cross-cloud scaling - on-premises data
- Identity architecture design
- Customer 360 with Azure Synapse and Dynamics 365 Customer Insights

Updated Articles

- SWIFT's Alliance Messaging Hub (AMH) with Alliance Connect (#6313f457f)
- Magento e-commerce platform in Azure Kubernetes Service (#cd094d538)
- Stream processing with Databricks (#63ef47fc7)
- Build a real-time recommendation API on Azure (#447842379)
- Intelligent e-commerce product search engine (#bacf3790c)
- Protect APIs with Azure Application Gateway and Azure API Management (#6f273719b)
- Speech services (#09941baed)
- Camera selection for Azure IoT Edge vision AI (#cc5ff9493)
- Computer vision with Azure IoT Edge (#cc5ff9493)
- Machine learning in IoT Edge Vision (#cc5ff9493)
- Suggest content tags with NLP using deep learning (#eeb4d1f81)
- Pricing models for a multitenant solution (#af94529db)
- Real-time fraud detection (#ce2fecdfdf)

- Monitor a microservices app in AKS (#bcb227d09)
- N-tier architecture style (#4a1bad2c3)
- End-to-end governance in Azure (#d92ed29b2)

January 2022

New Articles

- Solutions for the facilities and real estate industries
- Microservices assessment and readiness
- Solutions for the automotive, mobility, and transportation industries
- Conditional Access architecture and personas
- Conditional Access design principles and dependencies
- Conditional Access framework and policies
- Conditional Access for Zero Trust
- Hybrid relay connection in Azure and Azure Stack Hub
- Solutions for the travel and hospitality industries
- Databases architecture design
- Edge Workload Configuration pattern
- Storage design
- Automated API deployments using APIOps
- DevOps with Azure Stack Hub
- SaaS digital business journey on Azure
- Solutions for the game development industry
- Manage data across Azure SQL estate with Azure Purview
- Architectural approaches for governance and compliance in multitenant solutions
- DataOps checklist
- Data management across Azure Data Lake with Azure Purview
- DevSecOps with a rolling main branching strategy
- Use Azure Firewall to help protect an AKS cluster
- Azure NAT Gateway considerations for multitenancy
- Teacher-provisioned virtual labs in Azure
- Automate document processing by using Azure Form Recognizer
- SWIFT's Alliance Access in Azure
- SWIFT's Alliance Messaging Hub (AMH) with Alliance Connect
- Loosely coupled quantum computing
- Tightly coupled quantum computing
- DevOps for quantum computing
- CI/CD for quantum computing jobs
- Azure digital twins builder
- Deploy machine learning models to multiple lines
- Automating diagnostic Jupyter Notebook execution

Updated Articles

- DevOps checklist (#8d2302d7b)
- Azure resource organization in multitenant solutions (#0bc7de6d9)
- Asynchronous messaging options (#943e03c78)
- Analytics end-to-end with Azure Synapse (#c5e5aed19)

- [Azure Virtual Desktop for the enterprise \(#79adc0725\)](#)
- [SAP deployment on Azure using an Oracle database \(#f5c527723\)](#)
- [Valet Key pattern \(#7a9472004\)](#)
- [Computer vision with Azure IoT Edge \(#c79dfeaa5\)](#)
- [SAP HANA for Linux VMs in scale-up systems \(#b55c5fd90\)](#)
- [CI/CD pipeline for container-based workloads \(#bec078aad\)](#)
- [Run reservoir simulation software on Azure \(#bec078aad\)](#)
- [Scalable and secure WordPress on Azure \(#bec078aad\)](#)
- [Multiple forests with AD DS, Azure AD, and Azure AD DS \(#bec078aad\)](#)
- [Azure Stack remote offices and branches \(#bec078aad\)](#)
- [Disaster recovery for Azure Stack Hub VMs \(#bec078aad\)](#)
- [Batch scoring of Python models on Azure \(#bec078aad\)](#)
- [Basic web application \(#bec078aad\)](#)
- [Scalable web application \(#bec078aad\)](#)
- [On-premises network using ExpressRoute \(#bec078aad\)](#)
- [Troubleshoot a hybrid VPN connection \(#bec078aad\)](#)
- [Create a Gridwich cloud environment \(#bec078aad\)](#)
- [Gridwich content protection and DRM \(#bec078aad\)](#)
- [Logging in Gridwich \(#bec078aad\)](#)
- [Gridwich request-response messages \(#bec078aad\)](#)
- [Gridwich project naming and namespaces \(#bec078aad\)](#)
- [Gridwich operations for Azure Storage \(#bec078aad\)](#)
- [Gridwich keys and secrets management \(#bec078aad\)](#)
- [Gridwich Media Services setup and scaling \(#bec078aad\)](#)
- [Test Media Services V3 encoding \(#bec078aad\)](#)
- [Gridwich variable flow \(#bec078aad\)](#)
- [Cross-platform chat \(#bec078aad\)](#)
- [Data streaming with AKS \(#bec078aad\)](#)
- [Energy supply optimization \(#bec078aad\)](#)
- [IoT analytics with Azure Data Explorer \(#bec078aad\)](#)
- [Loan chargeoff prediction with SQL Server \(#bec078aad\)](#)
- [Run Oracle databases on Azure \(#bec078aad\)](#)
- [Retail and e-commerce using Azure MySQL \(#bec078aad\)](#)
- [Retail and e-commerce using Azure PostgreSQL \(#bec078aad\)](#)
- [Scalable web apps with Azure Redis Cache \(#bec078aad\)](#)
- [SAP workload development and test settings \(#c92dec5a3\)](#)
- [Secure research environment for regulated data \(#90428ee6b\)](#)
- [Baseline architecture for an AKS cluster \(#bb571243f\)](#)
- [AKS regulated cluster for PCI-DSS 3.2.1 - Network segmentation \(#60aaf99a3\)](#)
- [Security and identity with Azure and AWS \(#9614f75ff\)](#)
- [Automate multistage DevOps pipelines with Azure Pipelines \(#9614f75ff\)](#)

December 2021

New Articles

- [Automate multistage DevOps pipelines with Azure Pipelines](#)
- [Data analysis for regulated industries](#)

- Governance of Teams guest users
- Choose an Azure multiparty computing service
- Azure resource organization in multitenant solutions
- Application data protection for AKS workloads on Azure NetApp Files
- SQL Managed Instance with CMK
- Enhanced-security hybrid messaging — client access
- Enhanced-security hybrid messaging — mobile access
- Enhanced-security hybrid messaging — web access
- Extract text from objects using Power Automate and AI Builder
- Azure files secured by AD DS
- Architectural approaches for networking in multitenant solutions
- AI-based footfall detection
- Update route tables by using Azure Route Server
- Manage Microsoft 365 tenant configuration with Azure DevOps
- Enterprise file shares with disaster recovery
- Blue/green app deployments
- Solutions for the energy and environment industries
- Azure App Service and Azure Functions considerations for multitenancy
- Azure Machine Learning architecture
- Enhanced customer dimension with Dynamics 365 Customer Insights
- Scalable cloud applications and SRE
- Network-hardened web app
- CI/CD for Windows desktop apps

Updated Articles

- Architecting multitenant solutions on Azure (#7ea4a61ae)
- Service-specific guidance for a multitenant solution (#7ea4a61ae)
- Modeling stage of the Team Data Science Process lifecycle (#7c8304f17)
- Azure Virtual Desktop for the enterprise (#43a9e9bb1)
- Monitor hybrid availability, performance (#95f2e26b8)
- Consumer health portal on Azure (#9f23b73b8)
- Analytics end-to-end with Azure Synapse (#faaf3d26b)
- Feature engineering in machine learning (#ea2e85f88)
- Scalable order processing (#861fef949)
- Getting started with Azure IoT solutions (#28785d0ac)
- SAP system on Oracle Database on Azure (#bbd15b5f6)
- Securely managed web applications (#ac2473f81)
- E-commerce front end (#1e6701dea)
- Baseball decision analysis with ML.NET and Blazor (#55d8edcf0)
- Campaign optimization with SQL Server (#46c56a1d4)
- Retail and e-commerce using Cosmos DB (#46c56a1d4)
- Retail - Buy online, pick-up in store (BOPIS) (#5f755f231)
- Extend your on-premises big data investments with HDInsight (#4cf4dca0c)

Azure application architecture fundamentals

3/10/2022 • 3 minutes to read • [Edit Online](#)

This library of content presents a structured approach for designing applications on Azure that are scalable, secure, resilient, and highly available. The guidance is based on proven practices that we have learned from customer engagements.

Introduction

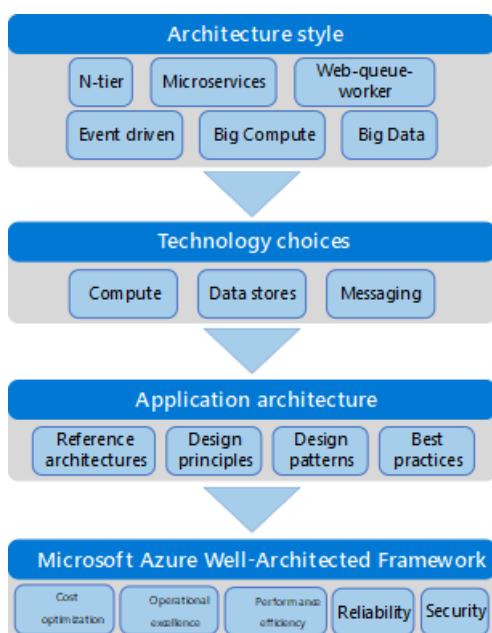
The cloud is changing how applications are designed and secured. Instead of monoliths, applications are decomposed into smaller, decentralized services. These services communicate through APIs or by using asynchronous messaging or eventing. Applications scale horizontally, adding new instances as demand requires.

These trends bring new challenges. Application states are distributed. Operations are done in parallel and asynchronously. Applications must be resilient when failures occur. Malicious actors continuously target applications. Deployments must be automated and predictable. Monitoring and telemetry are critical for gaining insight into the system. This guide is designed to help you navigate these changes.

TRADITIONAL ON-PREMISES	MODERN CLOUD
Monolithic Designed for predictable scalability Relational database Synchronized processing Design to avoid failures (MTBF) Occasional large updates Manual management Snowflake servers	Decomposed Designed for elastic scale Polyglot persistence (mix of storage technologies) Asynchronous processing Design for failure (MTTR) Frequent small updates Automated self-management Immutable infrastructure

How this guidance is structured

The Azure application architecture fundamentals guidance is organized as a series of steps, from the architecture and design to implementation. For each step, there is supporting guidance that will help you with the design of your application architecture.



Architecture styles

The first decision point is the most fundamental. What kind of architecture are you building? It might be a microservices architecture, a more traditional N-tier application, or a big data solution. We have identified several distinct architecture styles. There are benefits and challenges to each.

Learn more: [Architecture styles](#)

Technology choices

Knowing the type of architecture you are building, now you can start to choose the main technology pieces for the architecture. The following technology choices are critical:

- *Compute* refers to the hosting model for the computing resources that your applications run on. For more information, see [Choose a compute service](#).
- *Data stores* include databases but also storage for message queues, caches, logs, and anything else that an application might persist to storage. For more information, see [Choose a data store](#).
- *Messaging* technologies enable asynchronous messages between components of the system. For more information, see [Choose a messaging service](#).

You will probably have to make additional technology choices along the way, but these three elements (compute, data, and messaging) are central to most cloud applications and will determine many aspects of your design.

Design the architecture

Once you have chosen the architecture style and the major technology components, you are ready to tackle the specific design of your application. Every application is different, but the following resources can help you along the way:

Reference architectures

Depending on your scenario, one of our [reference architectures](#) may be a good starting point. Each reference architecture includes recommended practices, along with considerations for scalability, availability, security, resilience, and other aspects of the design. Most also include a deployable solution or reference implementation.

Design principles

We have identified 10 high-level design principles that will make your application more scalable, resilient, and manageable. These design principles apply to any architecture style. Throughout the design process, keep these 10 high-level design principles in mind. For more information, see [Design principles](#).

Design patterns

Software design patterns are repeatable patterns that are proven to solve specific problems. Our catalog of Cloud design patterns addresses specific challenges in distributed systems. They address aspects such as availability, high availability, operational excellence, resiliency, performance, and security. You can find our catalog of design patterns [here](#).

Best practices

Our [best practices](#) articles cover various design considerations including API design, autoscaling, data partitioning, caching, and so forth. Review these and apply the best practices that are appropriate for your application.

Security best practices

Our [security best practices](#) describe how to ensure that the confidentiality, integrity, and availability of your application aren't compromised by malicious actors.

Quality pillars

A successful cloud application will focus on five pillars of software quality: Cost optimization, Operational excellence, Performance efficiency, Reliability, and Security.

Leverage the [Microsoft Azure Well-Architected Framework](#) to assess your architecture across these five pillars.

Next steps

[Architecture styles](#)

Architecture styles

3/10/2022 • 5 minutes to read • [Edit Online](#)

An *architecture style* is a family of architectures that share certain characteristics. For example, [N-tier](#) is a common architecture style. More recently, [microservice architectures](#) have started to gain favor. Architecture styles don't require the use of particular technologies, but some technologies are well-suited for certain architectures. For example, containers are a natural fit for microservices.

We have identified a set of architecture styles that are commonly found in cloud applications. The article for each style includes:

- A description and logical diagram of the style.
- Recommendations for when to choose this style.
- Benefits, challenges, and best practices.
- A recommended deployment using relevant Azure services.

A quick tour of the styles

This section gives a quick tour of the architecture styles that we've identified, along with some high-level considerations for their use. Read more details in the linked topics.

N-tier

[N-tier](#) is a traditional architecture for enterprise applications. Dependencies are managed by dividing the application into *layers* that perform logical functions, such as presentation, business logic, and data access. A layer can only call into layers that sit below it. However, this horizontal layering can be a liability. It can be hard to introduce changes in one part of the application without touching the rest of the application. That makes frequent updates a challenge, limiting how quickly new features can be added.

N-tier is a natural fit for migrating existing applications that already use a layered architecture. For that reason, N-tier is most often seen in infrastructure as a service (IaaS) solutions, or application that use a mix of IaaS and managed services.

Web-Queue-Worker

For a purely PaaS solution, consider a [Web-Queue-Worker](#) architecture. In this style, the application has a web front end that handles HTTP requests and a back-end worker that performs CPU-intensive tasks or long-running operations. The front end communicates to the worker through an asynchronous message queue.

Web-queue-worker is suitable for relatively simple domains with some resource-intensive tasks. Like N-tier, the architecture is easy to understand. The use of managed services simplifies deployment and operations. But with complex domains, it can be hard to manage dependencies. The front end and the worker can easily become large, monolithic components that are hard to maintain and update. As with N-tier, this can reduce the frequency of updates and limit innovation.

Microservices

If your application has a more complex domain, consider moving to a [Microservices](#) architecture. A microservices application is composed of many small, independent services. Each service implements a single business capability. Services are loosely coupled, communicating through API contracts.

Each service can be built by a small, focused development team. Individual services can be deployed without a lot of coordination between teams, which encourages frequent updates. A microservice architecture is more complex to build and manage than either N-tier or web-queue-worker. It requires a mature development and

DevOps culture. But done right, this style can lead to higher release velocity, faster innovation, and a more resilient architecture.

Event-driven architecture

Event-Driven Architectures use a publish-subscribe (pub-sub) model, where producers publish events, and consumers subscribe to them. The producers are independent from the consumers, and consumers are independent from each other.

Consider an event-driven architecture for applications that ingest and process a large volume of data with very low latency, such as IoT solutions. The style is also useful when different subsystems must perform different types of processing on the same event data.

Big Data, Big Compute

Big Data and **Big Compute** are specialized architecture styles for workloads that fit certain specific profiles.

Big data divides a very large dataset into chunks, performing parallel processing across the entire set, for analysis and reporting. Big compute, also called high-performance computing (HPC), makes parallel computations across a large number (thousands) of cores. Domains include simulations, modeling, and 3-D rendering.

Architecture styles as constraints

An architecture style places constraints on the design, including the set of elements that can appear and the allowed relationships between those elements. Constraints guide the "shape" of an architecture by restricting the universe of choices. When an architecture conforms to the constraints of a particular style, certain desirable properties emerge.

For example, the constraints in microservices include:

- A service represents a single responsibility.
- Every service is independent of the others.
- Data is private to the service that owns it. Services do not share data.

By adhering to these constraints, what emerges is a system where services can be deployed independently, faults are isolated, frequent updates are possible, and it's easy to introduce new technologies into the application.

Before choosing an architecture style, make sure that you understand the underlying principles and constraints of that style. Otherwise, you can end up with a design that conforms to the style at a superficial level, but does not achieve the full potential of that style. It's also important to be pragmatic. Sometimes it's better to relax a constraint, rather than insist on architectural purity.

The following table summarizes how each style manages dependencies, and the types of domain that are best suited for each.

ARCHITECTURE STYLE	DEPENDENCY MANAGEMENT	DOMAIN TYPE
N-tier	Horizontal tiers divided by subnet	Traditional business domain. Frequency of updates is low.
Web-Queue-Worker	Front and backend jobs, decoupled by async messaging.	Relatively simple domain with some resource intensive tasks.
Microservices	Vertically (functionally) decomposed services that call each other through APIs.	Complicated domain. Frequent updates.

ARCHITECTURE STYLE	DEPENDENCY MANAGEMENT	DOMAIN TYPE
Event-driven architecture.	Producer/consumer. Independent view per sub-system.	IoT and real-time systems
Big data	Divide a huge dataset into small chunks. Parallel processing on local datasets.	Batch and real-time data analysis. Predictive analysis using ML.
Big compute	Data allocation to thousands of cores.	Compute intensive domains such as simulation.

Consider challenges and benefits

Constraints also create challenges, so it's important to understand the trade-offs when adopting any of these styles. Do the benefits of the architecture style outweigh the challenges, *for this subdomain and bounded context*.

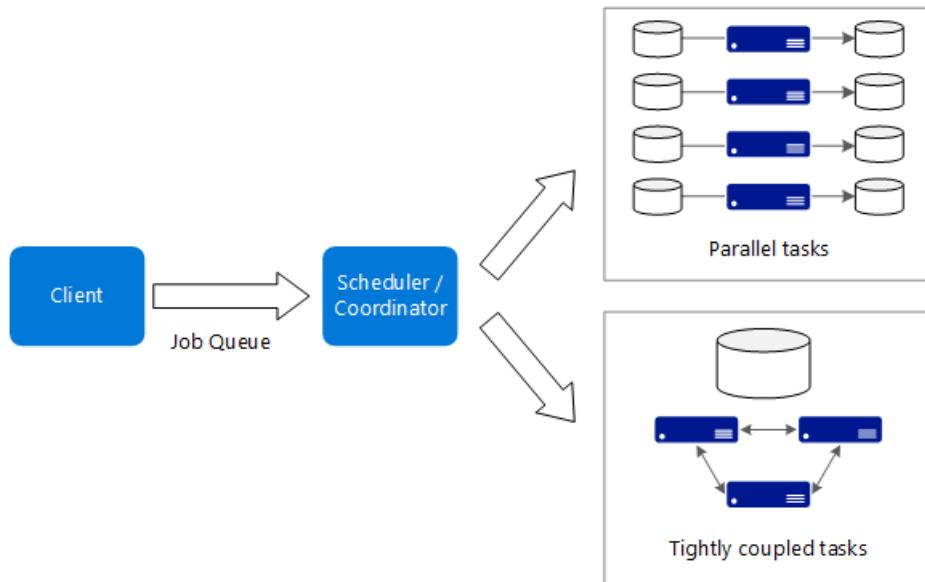
Here are some of the types of challenges to consider when selecting an architecture style:

- **Complexity.** Is the complexity of the architecture justified for your domain? Conversely, is the style too simplistic for your domain? In that case, you risk ending up with a "[big ball of mud](#)", because the architecture does not help you to manage dependencies cleanly.
- **Asynchronous messaging and eventual consistency.** Asynchronous messaging can be used to decouple services, and increase reliability (because messages can be retried) and scalability. However, this also creates challenges in handling eventual consistency, as well as the possibility of duplicate messages.
- **Inter-service communication.** As you decompose an application into separate services, there is a risk that communication between services will cause unacceptable latency or create network congestion (for example, in a microservices architecture).
- **Manageability.** How hard is it to manage the application, monitor, deploy updates, and so on?

Big compute architecture style

3/10/2022 • 3 minutes to read • [Edit Online](#)

The term *big compute* describes large-scale workloads that require a large number of cores, often numbering in the hundreds or thousands. Scenarios include image rendering, fluid dynamics, financial risk modeling, oil exploration, drug design, and engineering stress analysis, among others.



Here are some typical characteristics of big compute applications:

- The work can be split into discrete tasks, which can be run across many cores simultaneously.
- Each task is finite. It takes some input, does some processing, and produces output. The entire application runs for a finite amount of time (minutes to days). A common pattern is to provision a large number of cores in a burst, and then spin down to zero once the application completes.
- The application does not need to stay up 24/7. However, the system must handle node failures or application crashes.
- For some applications, tasks are independent and can run in parallel. In other cases, tasks are tightly coupled, meaning they must interact or exchange intermediate results. In that case, consider using high-speed networking technologies such as InfiniBand and remote direct memory access (RDMA).
- Depending on your workload, you might use compute-intensive VM sizes (H16r, H16mr, and A9).

When to use this architecture

- Computationally intensive operations such as simulation and number crunching.
- Simulations that are computationally intensive and must be split across CPUs in multiple computers (10-1000s).
- Simulations that require too much memory for one computer, and must be split across multiple computers.
- Long-running computations that would take too long to complete on a single computer.
- Smaller computations that must be run 100s or 1000s of times, such as Monte Carlo simulations.

Benefits

- High performance with "[embarrassingly parallel](#)" processing.
- Can harness hundreds or thousands of computer cores to solve large problems faster.

- Access to specialized high-performance hardware, with dedicated high-speed InfiniBand networks.
- You can provision VMs as needed to do work, and then tear them down.

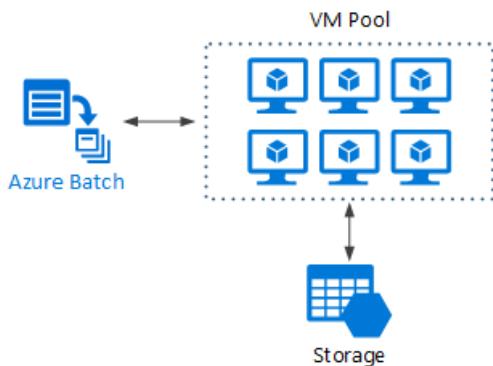
Challenges

- Managing the VM infrastructure.
- Managing the volume of number crunching
- Provisioning thousands of cores in a timely manner.
- For tightly coupled tasks, adding more cores can have diminishing returns. You may need to experiment to find the optimum number of cores.

Big compute using Azure Batch

[Azure Batch](#) is a managed service for running large-scale high-performance computing (HPC) applications.

Using Azure Batch, you configure a VM pool, and upload the applications and data files. Then the Batch service provisions the VMs, assign tasks to the VMs, runs the tasks, and monitors the progress. Batch can automatically scale out the VMs in response to the workload. Batch also provides job scheduling.

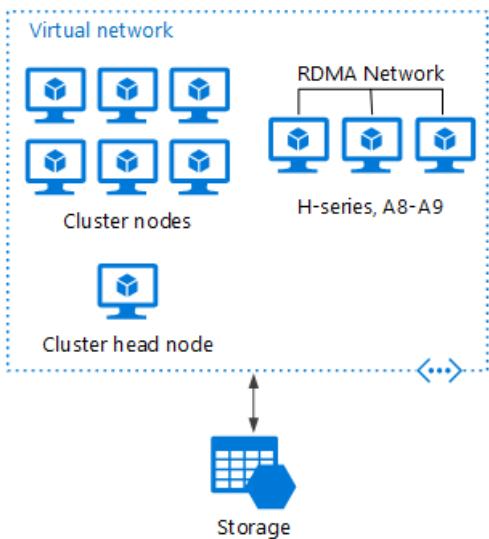


Big compute running on Virtual Machines

You can use [Microsoft HPC Pack](#) to administer a cluster of VMs, and schedule and monitor HPC jobs. With this approach, you must provision and manage the VMs and network infrastructure. Consider this approach if you have existing HPC workloads and want to move some or all it to Azure. You can move the entire HPC cluster to Azure, or you can keep your HPC cluster on-premises but use Azure for burst capacity. For more information, see [Batch and HPC solutions for large-scale computing workloads](#).

HPC Pack deployed to Azure

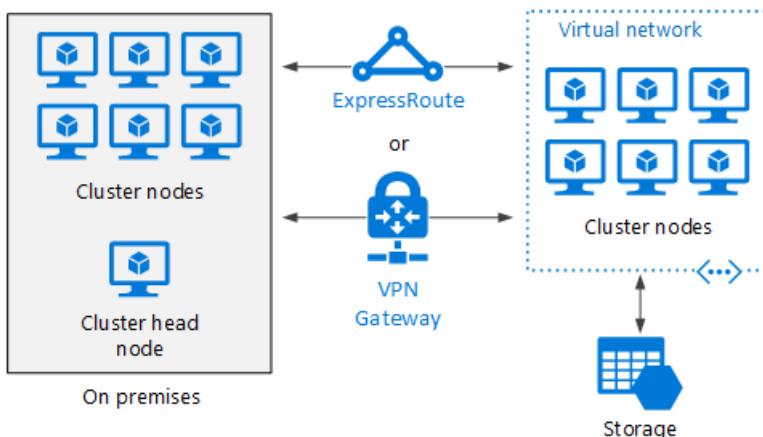
In this scenario, the HPC cluster is created entirely within Azure.



The head node provides management and job scheduling services to the cluster. For tightly coupled tasks, use an RDMA network that provides very high bandwidth, low latency communication between VMs. For more information, see [Deploy an HPC Pack 2016 cluster in Azure](#).

Burst an HPC cluster to Azure

In this scenario, an organization is running HPC Pack on-premises, and uses Azure VMs for burst capacity. The cluster head node is on-premises. ExpressRoute or VPN Gateway connects the on-premises network to the Azure VNet.



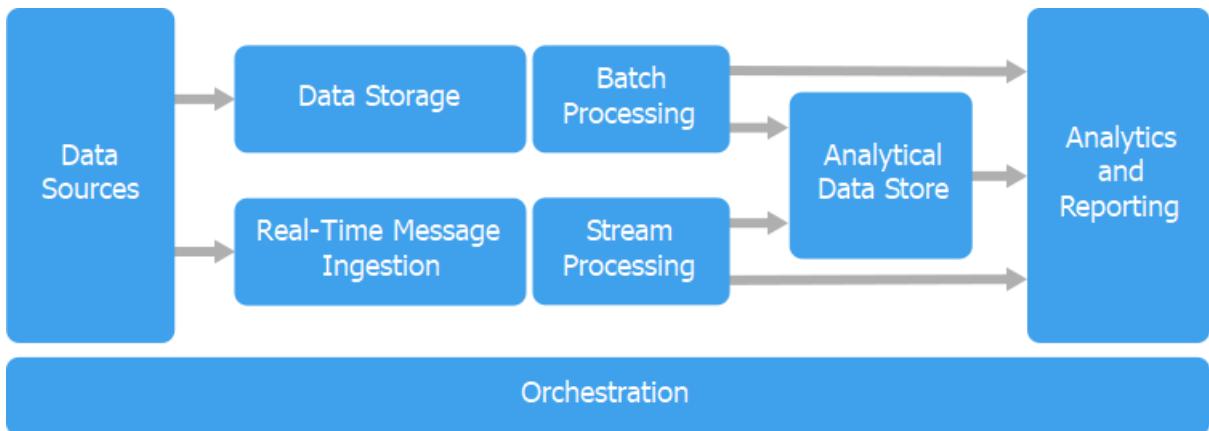
Next steps

- [Choose an Azure compute service for your application](#)
- [High Performance Computing \(HPC\) on Azure](#)
- [HPC cluster deployed in the cloud](#)

Big data architecture style

3/10/2022 • 10 minutes to read • [Edit Online](#)

A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



Big data solutions typically involve one or more of the following types of workload:

- Batch processing of big data sources at rest.
- Real-time processing of big data in motion.
- Interactive exploration of big data.
- Predictive analytics and machine learning.

Most big data architectures include some or all of the following components:

- **Data sources:** All big data solutions start with one or more data sources. Examples include:
 - Application data stores, such as relational databases.
 - Static files produced by applications, such as web server log files.
 - Real-time data sources, such as IoT devices.
- **Data storage:** Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats. This kind of store is often called a *data lake*. Options for implementing this storage include Azure Data Lake Store or blob containers in Azure Storage.
- **Batch processing:** Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files. Options include running U-SQL jobs in Azure Data Lake Analytics, using Hive, Pig, or custom Map/Reduce jobs in an HDInsight Hadoop cluster, or using Java, Scala, or Python programs in an HDInsight Spark cluster.
- **Real-time message ingestion:** If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing. This might be a simple data store, where incoming messages are dropped into a folder for processing. However, many solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics. Options include Azure Event Hubs, Azure IoT Hubs, and Kafka.
- **Stream processing:** After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink. Azure Stream Analytics provides a managed stream processing service based on

perpetually running SQL queries that operate on unbounded streams. You can also use open source Apache streaming technologies like Storm and Spark Streaming in an HDInsight cluster.

- **Analytical data store:** Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions. Alternatively, the data could be presented through a low-latency NoSQL technology such as HBase, or an interactive Hive database that provides a metadata abstraction over data files in the distributed data store. Azure Synapse Analytics provides a managed service for large-scale, cloud-based data warehousing. HDInsight supports Interactive Hive, HBase, and Spark SQL, which can also be used to serve data for analysis.
- **Analysis and reporting:** The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modeling layer, such as a multidimensional OLAP cube or tabular data model in Azure Analysis Services. It might also support self-service BI, using the modeling and visualization technologies in Microsoft Power BI or Microsoft Excel. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts. For these scenarios, many Azure services support analytical notebooks, such as Jupyter, enabling these users to leverage their existing skills with Python or R. For large-scale data exploration, you can use Microsoft R Server, either standalone or with Spark.
- **Orchestration:** Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such as Azure Data Factory or Apache Oozie and Sqoop.

Azure includes many services that can be used in a big data architecture. They fall roughly into two categories:

- Managed services, including Azure Data Lake Store, Azure Data Lake Analytics, Azure Synapse Analytics, Azure Stream Analytics, Azure Event Hub, Azure IoT Hub, and Azure Data Factory.
- Open source technologies based on the Apache Hadoop platform, including HDFS, HBase, Hive, Pig, Spark, Storm, Oozie, Sqoop, and Kafka. These technologies are available on Azure in the Azure HDInsight service.

These options are not mutually exclusive, and many solutions combine open source technologies with Azure services.

When to use this architecture

Consider this architecture style when you need to:

- Store and process data in volumes too large for a traditional database.
- Transform unstructured data for analysis and reporting.
- Capture, process, and analyze unbounded streams of data in real time, or with low latency.
- Use Azure Machine Learning or Microsoft Cognitive Services.

Benefits

- **Technology choices.** You can mix and match Azure managed services and Apache technologies in HDInsight clusters, to capitalize on existing skills or technology investments.
- **Performance through parallelism.** Big data solutions take advantage of parallelism, enabling high-performance solutions that scale to large volumes of data.
- **Elastic scale.** All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.

- **Interoperability with existing solutions.** The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

Challenges

- **Complexity.** Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes. Moreover, there may be a large number of configuration settings across multiple systems that must be used in order to optimize performance.
- **Skillset.** Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages. For example, the U-SQL language in Azure Data Lake Analytics is based on a combination of Transact-SQL and C#. Similarly, SQL-based APIs are available for Hive, HBase, and Spark.
- **Technology maturity.** Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release. Managed services such as Azure Data Lake Analytics and Azure Data Factory are relatively young, compared with other Azure services, and will likely evolve over time.
- **Security.** Big data solutions usually rely on storing all static data in a centralized data lake. Securing access to this data can be challenging, especially when the data must be ingested and consumed by multiple applications and platforms.

Best practices

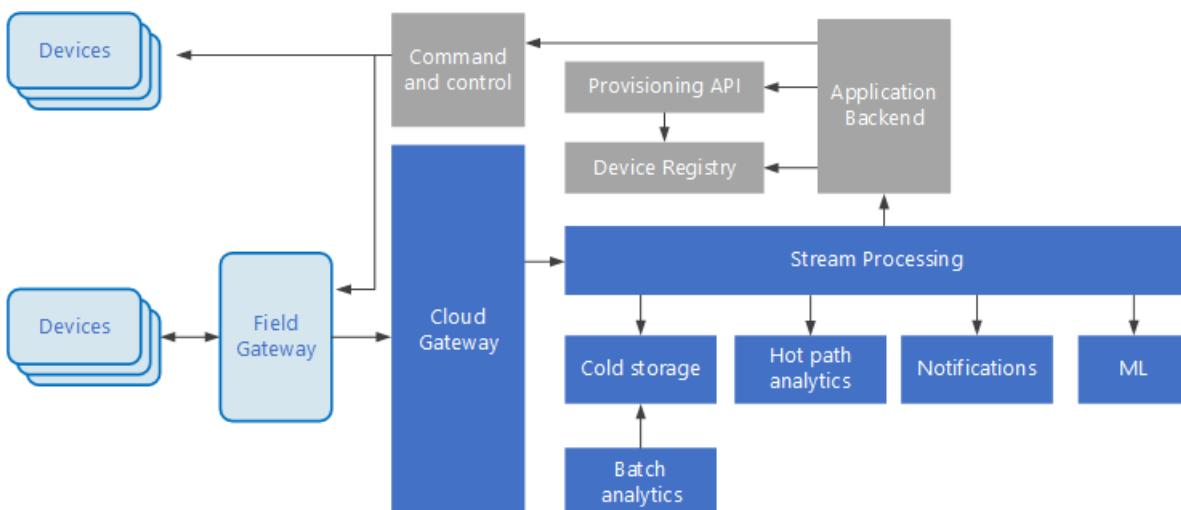
- **Leverage parallelism.** Most big data processing technologies distribute the workload across multiple processing units. This requires that static data files are created and stored in a splittable format. Distributed file systems such as HDFS can optimize read and write performance, and the actual processing is performed by multiple cluster nodes in parallel, which reduces overall job times.
- **Partition data.** Batch processing usually happens on a recurring schedule — for example, weekly or monthly. Partition data files, and data structures such as tables, based on temporal periods that match the processing schedule. That simplifies data ingestion and job scheduling, and makes it easier to troubleshoot failures. Also, partitioning tables that are used in Hive, U-SQL, or SQL queries can significantly improve query performance.
- **Apply schema-on-read semantics.** Using a data lake lets you to combine storage for files in multiple formats, whether structured, semi-structured, or unstructured. Use *schema-on-read* semantics, which project a schema onto the data when the data is processing, not when the data is stored. This builds flexibility into the solution, and prevents bottlenecks during data ingestion caused by data validation and type checking.
- **Process data in-place.** Traditional BI solutions often use an extract, transform, and load (ETL) process to move data into a data warehouse. With larger volumes data, and a greater variety of formats, big data solutions generally use variations of ETL, such as transform, extract, and load (TEL). With this approach, the data is processed within the distributed data store, transforming it to the required structure, before moving the transformed data into an analytical data store.
- **Balance utilization and time costs.** For batch processing jobs, it's important to consider two factors: The per-unit cost of the compute nodes, and the per-minute cost of using those nodes to complete the job. For example, a batch job may take eight hours with four cluster nodes. However, it might turn out that the job uses all four nodes only during the first two hours, and after that, only two nodes are required. In that case, running the entire job on two nodes would increase the total job time, but would not double it, so the total cost would be less. In some business scenarios, a longer processing time may

be preferable to the higher cost of using underutilized cluster resources.

- **Separate cluster resources.** When deploying HDInsight clusters, you will normally achieve better performance by provisioning separate cluster resources for each type of workload. For example, although Spark clusters include Hive, if you need to perform extensive processing with both Hive and Spark, you should consider deploying separate dedicated Spark and Hadoop clusters. Similarly, if you are using HBase and Storm for low latency stream processing and Hive for batch processing, consider separate clusters for Storm, HBase, and Hadoop.
- **Orchestrate data ingestion.** In some cases, existing business applications may write data files for batch processing directly into Azure storage blob containers, where they can be consumed by HDInsight or Azure Data Lake Analytics. However, you will often need to orchestrate the ingestion of data from on-premises or external data sources into the data lake. Use an orchestration workflow or pipeline, such as those supported by Azure Data Factory or Oozie, to achieve this in a predictable and centrally manageable fashion.
- **Scrub sensitive data early.** The data ingestion workflow should scrub sensitive data early in the process, to avoid storing it in the data lake.

IoT architecture

Internet of Things (IoT) is a specialized subset of big data solutions. The following diagram shows a possible logical architecture for IoT. The diagram emphasizes the event-streaming components of the architecture.



The **cloud gateway** ingests device events at the cloud boundary, using a reliable, low latency messaging system.

Devices might send events directly to the cloud gateway, or through a **field gateway**. A field gateway is a specialized device or software, usually colocated with the devices, that receives events and forwards them to the cloud gateway. The field gateway might also preprocess the raw device events, performing functions such as filtering, aggregation, or protocol transformation.

After ingestion, events go through one or more **stream processors** that can route the data (for example, to storage) or perform analytics and other processing.

The following are some common types of processing. (This list is certainly not exhaustive.)

- Writing event data to cold storage, for archiving or batch analytics.
- Hot path analytics, analyzing the event stream in (near) real time, to detect anomalies, recognize patterns over rolling time windows, or trigger alerts when a specific condition occurs in the stream.
- Handling special types of non-telemetry messages from devices, such as notifications and alarms.

- Machine learning.

The boxes that are shaded gray show components of an IoT system that are not directly related to event streaming, but are included here for completeness.

- The **device registry** is a database of the provisioned devices, including the device IDs and usually device metadata, such as location.
- The **provisioning API** is a common external interface for provisioning and registering new devices.
- Some IoT solutions allow **command and control messages** to be sent to devices.

This section has presented a very high-level view of IoT, and there are many subtleties and challenges to consider. For a more detailed reference architecture and discussion, see the [Microsoft Azure IoT Reference Architecture](#) (PDF download).

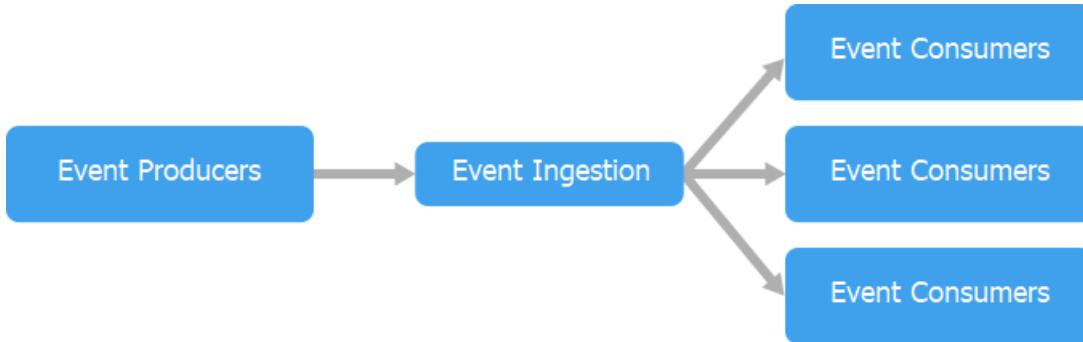
Next steps

- Learn more about [big data architectures](#).
- Learn more about [IoT solutions](#).

Event-driven architecture style

3/10/2022 • 3 minutes to read • [Edit Online](#)

An event-driven architecture consists of **event producers** that generate a stream of events, and **event consumers** that listen for the events.



Events are delivered in near real time, so consumers can respond immediately to events as they occur. Producers are decoupled from consumers — a producer doesn't know which consumers are listening. Consumers are also decoupled from each other, and every consumer sees all of the events. This differs from a [Competing Consumers](#) pattern, where consumers pull messages from a queue and a message is processed just once (assuming no errors). In some systems, such as IoT, events must be ingested at very high volumes.

An event driven architecture can use a pub/sub model or an event stream model.

- **Pub/sub:** The messaging infrastructure keeps track of subscriptions. When an event is published, it sends the event to each subscriber. After an event is received, it cannot be replayed, and new subscribers do not see the event.
- **Event streaming:** Events are written to a log. Events are strictly ordered (within a partition) and durable. Clients don't subscribe to the stream, instead a client can read from any part of the stream. The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

On the consumer side, there are some common variations:

- **Simple event processing.** An event immediately triggers an action in the consumer. For example, you could use Azure Functions with a Service Bus trigger, so that a function executes whenever a message is published to a Service Bus topic.
- **Complex event processing.** A consumer processes a series of events, looking for patterns in the event data, using a technology such as Azure Stream Analytics or Apache Storm. For example, you could aggregate readings from an embedded device over a time window, and generate a notification if the moving average crosses a certain threshold.
- **Event stream processing.** Use a data streaming platform, such as Azure IoT Hub or Apache Kafka, as a pipeline to ingest events and feed them to stream processors. The stream processors act to process or transform the stream. There may be multiple stream processors for different subsystems of the application. This approach is a good fit for IoT workloads.

The source of the events may be external to the system, such as physical devices in an IoT solution. In that case, the system must be able to ingest the data at the volume and throughput that is required by the data source.

In the logical diagram above, each type of consumer is shown as a single box. In practice, it's common to have

multiple instances of a consumer, to avoid having the consumer become a single point of failure in system. Multiple instances might also be necessary to handle the volume and frequency of events. Also, a single consumer might process events on multiple threads. This can create challenges if events must be processed in order or require exactly-once semantics. See [Minimize Coordination](#).

When to use this architecture

- Multiple subsystems must process the same events.
- Real-time processing with minimum time lag.
- Complex event processing, such as pattern matching or aggregation over time windows.
- High volume and high velocity of data, such as IoT.

Benefits

- Producers and consumers are decoupled.
- No point-to-point integrations. It's easy to add new consumers to the system.
- Consumers can respond to events immediately as they arrive.
- Highly scalable and distributed.
- Subsystems have independent views of the event stream.

Challenges

- Guaranteed delivery. In some systems, especially in IoT scenarios, it's crucial to guarantee that events are delivered.
- Processing events in order or exactly once. Each consumer type typically runs in multiple instances, for resiliency and scalability. This can create a challenge if the events must be processed in order (within a consumer type), or if the processing logic is not idempotent.

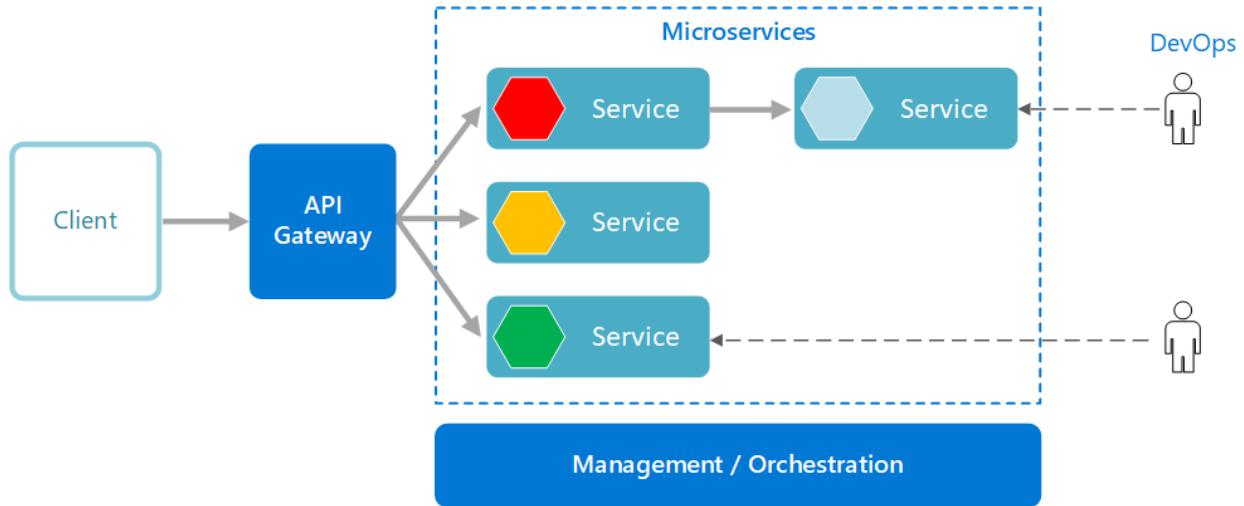
Additional considerations

- The amount of data to include in an event can be a significant consideration that affects both performance and cost. Putting all the relevant information needed for processing in the event itself can simplify the processing code and save additional lookups. Putting the minimal amount of information in an event, like just a couple of identifiers, will reduce transport time and cost, but requires the processing code to look up any additional information it needs. For more information on this, take a look at [this blog post](#).

Microservices architecture style

3/10/2022 • 6 minutes to read • [Edit Online](#)

A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability within a bounded context. A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.



What are microservices?

- Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Supports polyglot programming. For example, services don't need to share the same technology stack, libraries, or frameworks.

Besides for the services themselves, some other components appear in a typical microservices architecture:

Management/orchestration. This component is responsible for placing services on nodes, identifying failures, rebalancing services across nodes, and so forth. Typically this component is an off-the-shelf technology such as Kubernetes, rather than something custom built.

API Gateway. The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.

Advantages of using an API gateway include:

- It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.

- Services can use messaging protocols that are not web friendly, such as AMQP.
- The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
- Out-of-the-box policies, like for throttling, caching, transformation, or validation.

Benefits

- **Agility.** Because microservices are deployed independently, it's easier to manage bug fixes and feature releases. You can update a service without redeploying the entire application, and roll back an update if something goes wrong. In many traditional applications, if a bug is found in one part of the application, it can block the entire release process. New features may be held up waiting for a bug fix to be integrated, tested, and published.
- **Small, focused teams.** A microservice should be small enough that a single feature team can build, test, and deploy it. Small team sizes promote greater agility. Large teams tend to be less productive, because communication is slower, management overhead goes up, and agility diminishes.
- **Small code base.** In a monolithic application, there is a tendency over time for code dependencies to become tangled. Adding a new feature requires touching code in a lot of places. By not sharing code or data stores, a microservices architecture minimizes dependencies, and that makes it easier to add new features.
- **Mix of technologies.** Teams can pick the technology that best fits their service, using a mix of technology stacks as appropriate.
- **Fault isolation.** If an individual microservice becomes unavailable, it won't disrupt the entire application, as long as any upstream microservices are designed to handle faults correctly (for example, by implementing circuit breaking).
- **Scalability.** Services can be scaled independently, letting you scale out subsystems that require more resources, without scaling out the entire application. Using an orchestrator such as Kubernetes or Service Fabric, you can pack a higher density of services onto a single host, which allows for more efficient utilization of resources.
- **Data isolation.** It is much easier to perform schema updates, because only a single microservice is affected. In a monolithic application, schema updates can become very challenging, because different parts of the application may all touch the same data, making any alterations to the schema risky.

Challenges

The benefits of microservices don't come for free. Here are some of the challenges to consider before embarking on a microservices architecture.

- **Complexity.** A microservices application has more moving parts than the equivalent monolithic application. Each service is simpler, but the entire system as a whole is more complex.
- **Development and testing.** Writing a small service that relies on other dependent services requires a different approach than writing a traditional monolithic or layered application. Existing tools are not always designed to work with service dependencies. Refactoring across service boundaries can be difficult. It is also challenging to test service dependencies, especially when the application is evolving quickly.
- **Lack of governance.** The decentralized approach to building microservices has advantages, but it can also lead to problems. You may end up with so many different languages and frameworks that the application becomes hard to maintain. It may be useful to put some project-wide standards in place,

without overly restricting teams' flexibility. This especially applies to cross-cutting functionality such as logging.

- **Network congestion and latency.** The use of many small, granular services can result in more interservice communication. Also, if the chain of service dependencies gets too long (service A calls B, which calls C...), the additional latency can become a problem. You will need to design APIs carefully. Avoid overly chatty APIs, think about serialization formats, and look for places to use asynchronous communication patterns like [queue-based load leveling](#).
- **Data integrity.** With each microservice responsible for its own data persistence. As a result, data consistency can be a challenge. Embrace eventual consistency where possible.
- **Management.** To be successful with microservices requires a mature DevOps culture. Correlated logging across services can be challenging. Typically, logging must correlate multiple service calls for a single user operation.
- **Versioning.** Updates to a service must not break services that depend on it. Multiple services could be updated at any given time, so without careful design, you might have problems with backward or forward compatibility.
- **Skill set.** Microservices are highly distributed systems. Carefully evaluate whether the team has the skills and experience to be successful.

Best practices

- Model services around the business domain.
- Decentralize everything. Individual teams are responsible for designing and building services. Avoid sharing code or data schemas.
- Data storage should be private to the service that owns the data. Use the best storage for each service and data type.
- Services communicate through well-designed APIs. Avoid leaking implementation details. APIs should model the domain, not the internal implementation of the service.
- Avoid coupling between services. Causes of coupling include shared database schemas and rigid communication protocols.
- Offload cross-cutting concerns, such as authentication and SSL termination, to the gateway.
- Keep domain knowledge out of the gateway. The gateway should handle and route client requests without any knowledge of the business rules or domain logic. Otherwise, the gateway becomes a dependency and can cause coupling between services.
- Services should have loose coupling and high functional cohesion. Functions that are likely to change together should be packaged and deployed together. If they reside in separate services, those services end up being tightly coupled, because a change in one service will require updating the other service. Overly chatty communication between two services may be a symptom of tight coupling and low cohesion.
- Isolate failures. Use resiliency strategies to prevent failures within a service from cascading. See [Resiliency patterns](#) and [Designing reliable applications](#).

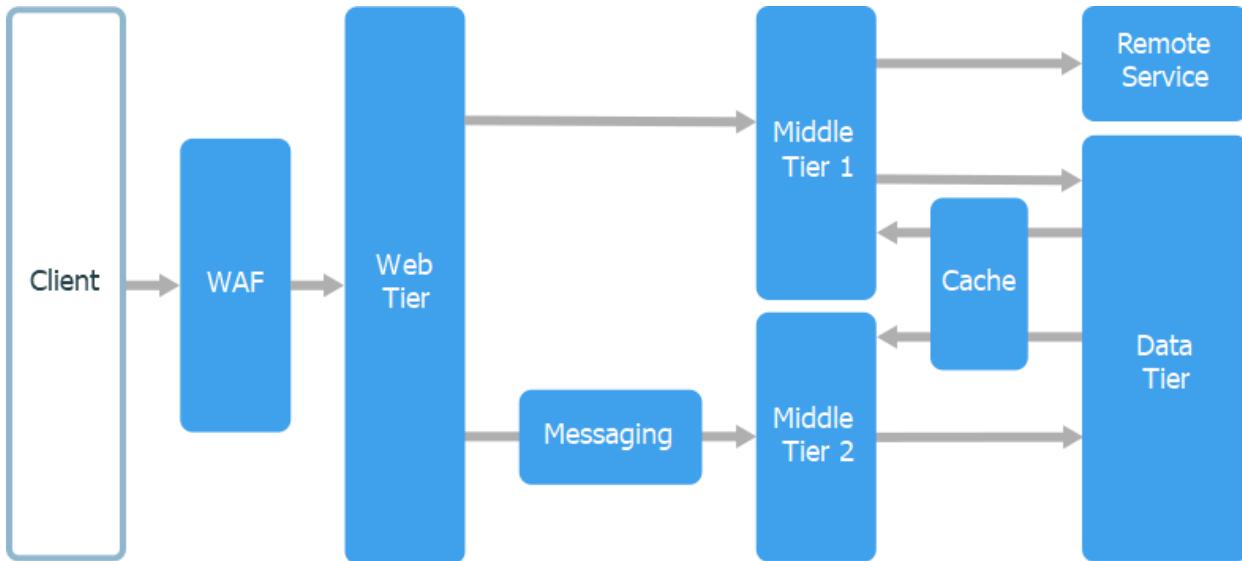
Next steps

For detailed guidance about building a microservices architecture on Azure, see [Designing, building, and operating microservices on Azure](#).

N-tier architecture style

3/10/2022 • 5 minutes to read • [Edit Online](#)

An N-tier architecture divides an application into **logical layers** and **physical tiers**.



Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility. A higher layer can use services in a lower layer, but not the other way around.

Tiers are physically separated, running on separate machines. A tier can call to another tier directly, or use asynchronous messaging (message queue). Although each layer might be hosted in its own tier, that's not required. Several layers might be hosted on the same tier. Physically separating the tiers improves scalability and resiliency, but also adds latency from the additional network communication.

A traditional three-tier application has a presentation tier, a middle tier, and a database tier. The middle tier is optional. More complex applications can have more than three tiers. The diagram above shows an application with two middle tiers, encapsulating different areas of functionality.

An N-tier application can have a **closed layer architecture** or an **open layer architecture**:

- In a closed layer architecture, a layer can only call the next layer immediately down.
- In an open layer architecture, a layer can call any of the layers below it.

A closed layer architecture limits the dependencies between layers. However, it might create unnecessary network traffic, if one layer simply passes requests along to the next layer.

When to use this architecture

N-tier architectures are typically implemented as infrastructure-as-service (IaaS) applications, with each tier running on a separate set of VMs. However, an N-tier application doesn't need to be pure IaaS. Often, it's advantageous to use managed services for some parts of the architecture, particularly caching, messaging, and data storage.

Consider an N-tier architecture for:

- Simple web applications.
- Migrating an on-premises application to Azure with minimal refactoring.
- Unified development of on-premises and cloud applications.

N-tier architectures are very common in traditional on-premises applications, so it's a natural fit for migrating existing workloads to Azure.

Benefits

- Portability between cloud and on-premises, and between cloud platforms.
- Less learning curve for most developers.
- Natural evolution from the traditional application model.
- Open to heterogeneous environment (Windows/Linux)

Challenges

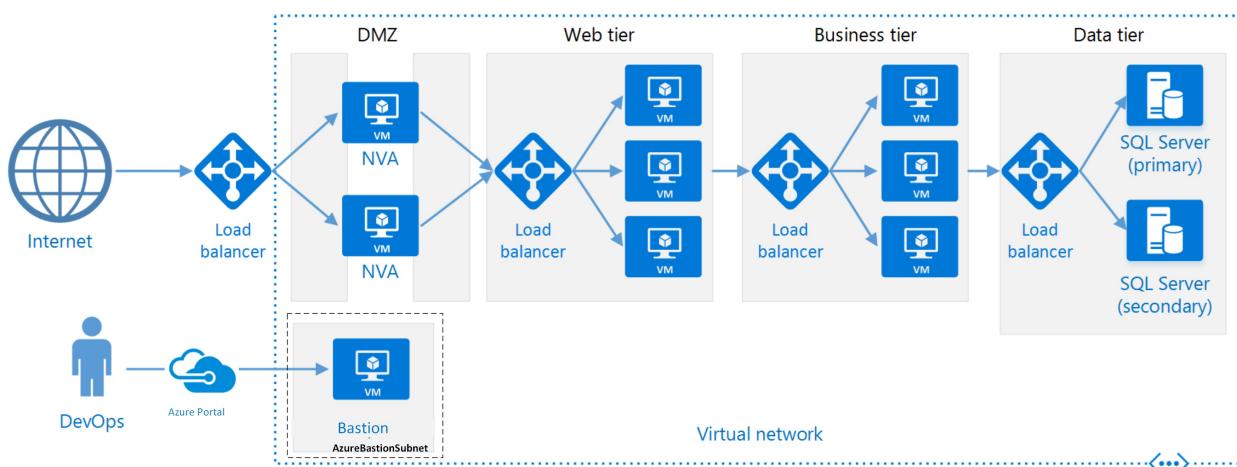
- It's easy to end up with a middle tier that just does CRUD operations on the database, adding extra latency without doing any useful work.
- Monolithic design prevents independent deployment of features.
- Managing an IaaS application is more work than an application that uses only managed services.
- It can be difficult to manage network security in a large system.

Best practices

- Use autoscaling to handle changes in load. See [Autoscaling best practices](#).
- Use [asynchronous messaging](#) to decouple tiers.
- Cache semistatic data. See [Caching best practices](#).
- Configure the database tier for high availability, using a solution such as [SQL Server Always On availability groups](#).
- Place a web application firewall (WAF) between the front end and the Internet.
- Place each tier in its own subnet, and use subnets as a security boundary.
- Restrict access to the data tier, by allowing requests only from the middle tier(s).

N-tier architecture on virtual machines

This section describes a recommended N-tier architecture running on VMs.



Each tier consists of two or more VMs, placed in an availability set or virtual machine scale set. Multiple VMs provide resiliency in case one VM fails. Load balancers are used to distribute requests across the VMs in a tier. A tier can be scaled horizontally by adding more VMs to the pool.

Each tier is also placed inside its own subnet, meaning their internal IP addresses fall within the same address range. That makes it easy to apply network security group rules and route tables to individual tiers.

The web and business tiers are stateless. Any VM can handle any request for that tier. The data tier should consist of a replicated database. For Windows, we recommend SQL Server, using Always On availability groups for high availability. For Linux, choose a database that supports replication, such as Apache Cassandra.

Network security groups restrict access to each tier. For example, the database tier only allows access from the business tier.

NOTE

The layer labeled "Business Tier" in our reference diagram is a moniker to the business logic tier. Likewise, we also call the presentation tier the "Web Tier." In our example, this is a web application, though multi-tier architectures can be used for other topologies as well (like desktop apps). Name your tiers what works best for your team to communicate the intent of that logical and/or physical tier in your application - you could even express that naming in resources you choose to represent that tier (e.g. vmss-appName-business-layer).

For more information about running N-tier applications on Azure:

- [Run Windows VMs for an N-tier application](#)
- [Windows N-tier application on Azure with SQL Server](#)
- [Microsoft Learn module: Tour the N-tier architecture style](#)
- [Azure Bastion](#)

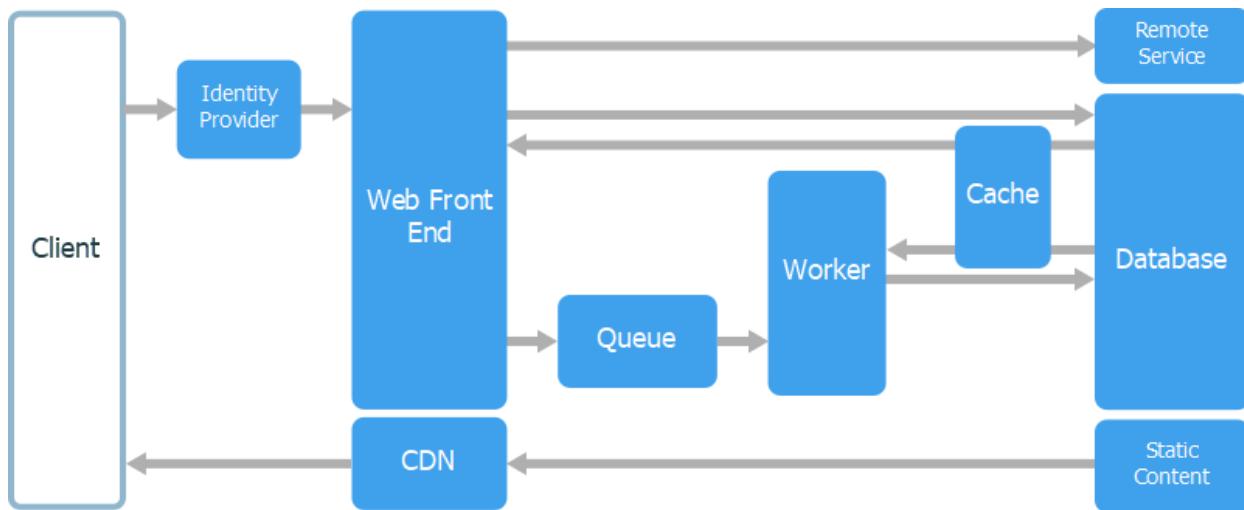
Additional considerations

- N-tier architectures are not restricted to three tiers. For more complex applications, it is common to have more tiers. In that case, consider using layer-7 routing to route requests to a particular tier.
- Tiers are the boundary of scalability, reliability, and security. Consider having separate tiers for services with different requirements in those areas.
- Use virtual machine scale sets for autoscaling.
- Look for places in the architecture where you can use a managed service without significant refactoring. In particular, look at caching, messaging, storage, and databases.
- For higher security, place a network DMZ in front of the application. The DMZ includes network virtual appliances (NVAs) that implement security functionality such as firewalls and packet inspection. For more information, see [Network DMZ reference architecture](#).
- For high availability, place two or more NVAs in an availability set, with an external load balancer to distribute Internet requests across the instances. For more information, see [Deploy highly available network virtual appliances](#).
- Do not allow direct RDP or SSH access to VMs that are running application code. Instead, operators should log into a jumpbox, also called a bastion host. This is a VM on the network that administrators use to connect to the other VMs. The jumpbox has a network security group that allows RDP or SSH only from approved public IP addresses.
- You can extend the Azure virtual network to your on-premises network using a site-to-site virtual private network (VPN) or Azure ExpressRoute. For more information, see [Hybrid network reference architecture](#).
- If your organization uses Active Directory to manage identity, you may want to extend your Active Directory environment to the Azure VNet. For more information, see [Identity management reference architecture](#).
- If you need higher availability than the Azure SLA for VMs provides, replicate the application across two regions and use Azure Traffic Manager for failover. For more information, see [Run Windows VMs in multiple regions](#) or [Run Linux VMs in multiple regions](#).

Web-Queue-Worker architecture style

3/10/2022 • 3 minutes to read • [Edit Online](#)

The core components of this architecture are a **web front end** that serves client requests, and a **worker** that performs resource-intensive tasks, long-running workflows, or batch jobs. The web front end communicates with the worker through a **message queue**.



Other components that are commonly incorporated into this architecture include:

- One or more databases.
- A cache to store values from the database for quick reads.
- CDN to serve static content
- Remote services, such as email or SMS service. Often these are provided by third parties.
- Identity provider for authentication.

The web and worker are both stateless. Session state can be stored in a distributed cache. Any long-running work is done asynchronously by the worker. The worker can be triggered by messages on the queue, or run on a schedule for batch processing. The worker is an optional component. If there are no long-running operations, the worker can be omitted.

The front end might consist of a web API. On the client side, the web API can be consumed by a single-page application that makes AJAX calls, or by a native client application.

When to use this architecture

The Web-Queue-Worker architecture is typically implemented using managed compute services, either Azure App Service or Azure Cloud Services.

Consider this architecture style for:

- Applications with a relatively simple domain.
- Applications with some long-running workflows or batch operations.
- When you want to use managed services, rather than infrastructure as a service (IaaS).

Benefits

- Relatively simple architecture that is easy to understand.

- Easy to deploy and manage.
- Clear separation of concerns.
- The front end is decoupled from the worker using asynchronous messaging.
- The front end and the worker can be scaled independently.

Challenges

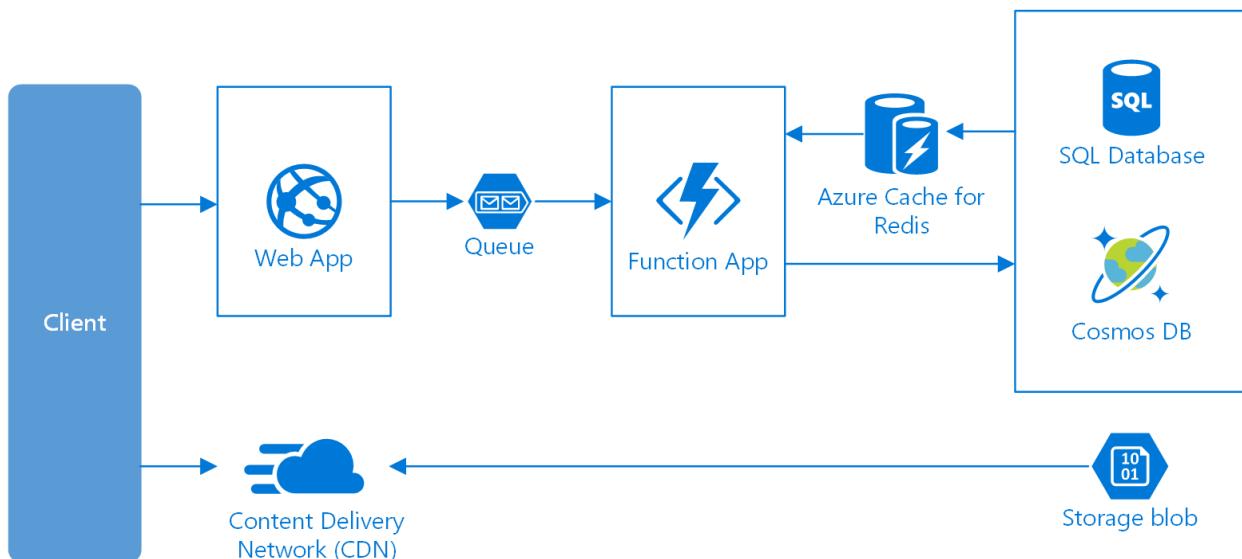
- Without careful design, the front end and the worker can become large, monolithic components that are difficult to maintain and update.
- There may be hidden dependencies, if the front end and worker share data schemas or code modules.

Best practices

- Expose a well-designed API to the client. See [API design best practices](#).
- Autoscale to handle changes in load. See [Autoscaling best practices](#).
- Cache semi-static data. See [Caching best practices](#).
- Use a CDN to host static content. See [CDN best practices](#).
- Use polyglot persistence when appropriate. See [Use the best data store for the job](#).
- Partition data to improve scalability, reduce contention, and optimize performance. See [Data partitioning best practices](#).

Web-Queue-Worker on Azure App Service

This section describes a recommended Web-Queue-Worker architecture that uses Azure App Service.



- The front end is implemented as an Azure App Service web app, and the worker is implemented as an Azure Functions app. The web app and the function app are both associated with an App Service plan that provides the VM instances.
- You can use either Azure Service Bus or Azure Storage queues for the message queue. (The diagram shows an Azure Storage queue.)
- Azure Cache for Redis stores session state and other data that needs low latency access.
- Azure CDN is used to cache static content such as images, CSS, or HTML.
- For storage, choose the storage technologies that best fit the needs of the application. You might use multiple storage technologies (polyglot persistence). To illustrate this idea, the diagram shows Azure SQL Database and Azure Cosmos DB.

For more details, see [App Service web application reference architecture](#).

Additional considerations

- Not every transaction has to go through the queue and worker to storage. The web front end can perform simple read/write operations directly. Workers are designed for resource-intensive tasks or long-running workflows. In some cases, you might not need a worker at all.
- Use the built-in autoscale feature of App Service to scale out the number of VM instances. If the load on the application follows predictable patterns, use schedule-based autoscale. If the load is unpredictable, use metrics-based autoscaling rules.
- Consider putting the web app and the function app into separate App Service plans. That way, they can be scaled independently.
- Use separate App Service plans for production and testing. Otherwise, if you use the same plan for production and testing, it means your tests are running on your production VMs.
- Use deployment slots to manage deployments. This lets you to deploy an updated version to a staging slot, then swap over to the new version. It also lets you swap back to the previous version, if there was a problem with the update.

Ten design principles for Azure applications

3/10/2022 • 2 minutes to read • [Edit Online](#)

Follow these design principles to make your application more scalable, resilient, and manageable.

Design for self healing. In a distributed system, failures happen. Design your application to be self healing when failures occur.

Make all things redundant. Build redundancy into your application, to avoid having single points of failure.

Minimize coordination. Minimize coordination between application services to achieve scalability.

Design to scale out. Design your application so that it can scale horizontally, adding or removing new instances as demand requires.

Partition around limits. Use partitioning to work around database, network, and compute limits.

Design for operations. Design your application so that the operations team has the tools they need.

Use managed services. When possible, use platform as a service (PaaS) rather than infrastructure as a service (IaaS).

Use the best data store for the job. Pick the storage technology that is the best fit for your data and how it will be used.

Design for evolution. All successful applications change over time. An evolutionary design is key for continuous innovation.

Build for the needs of business. Every design decision must be justified by a business requirement.

Design for self healing

3/10/2022 • 4 minutes to read • [Edit Online](#)

Design your application to be self healing when failures occur

In a distributed system, failures can happen. Hardware can fail. The network can have transient failures. Rarely, an entire service or region may experience a disruption, but even those must be planned for.

Therefore, design an application to be self healing when failures occur. This requires a three-pronged approach:

- Detect failures.
- Respond to failures gracefully.
- Log and monitor failures, to give operational insight.

How you respond to a particular type of failure may depend on your application's availability requirements. For example, if you require very high availability, you might automatically fail over to a secondary region during a regional outage. However, that will incur a higher cost than a single-region deployment.

Also, don't just consider big events like regional outages, which are generally rare. You should focus as much, if not more, on handling local, short-lived failures, such as network connectivity failures or failed database connections.

Recommendations

Retry failed operations. Transient failures may occur due to momentary loss of network connectivity, a dropped database connection, or a timeout when a service is busy. Build retry logic into your application to handle transient failures. For many Azure services, the client SDK implements automatic retries. For more information, see [Transient fault handling](#) and the [Retry pattern](#).

Protect failing remote services (Circuit Breaker). It's good to retry after a transient failure, but if the failure persists, you can end up with too many callers hammering a failing service. This can lead to cascading failures, as requests back up. Use the [Circuit Breaker pattern](#) to fail fast (without making the remote call) when an operation is likely to fail.

Isolate critical resources (Bulkhead). Failures in one subsystem can sometimes cascade. This can happen if a failure causes some resources, such as threads or sockets, not to get freed in a timely manner, leading to resource exhaustion. To avoid this, partition a system into isolated groups, so that a failure in one partition does not bring down the entire system.

Perform load leveling. Applications may experience sudden spikes in traffic that can overwhelm services on the backend. To avoid this, use the [Queue-Based Load Leveling pattern](#) to queue work items to run asynchronously. The queue acts as a buffer that smooths out peaks in the load.

Fail over. If an instance can't be reached, fail over to another instance. For things that are stateless, like a web server, put several instances behind a load balancer or traffic manager. For things that store state, like a database, use replicas and fail over. Depending on the data store and how it replicates, this may require the application to deal with eventual consistency.

Compensate failed transactions. In general, avoid distributed transactions, as they require coordination across services and resources. Instead, compose an operation from smaller individual transactions. If the operation fails midway through, use [Compensating Transactions](#) to undo any step that already completed.

Checkpoint long-running transactions. Checkpoints can provide resiliency if a long-running operation fails.

When the operation restarts (for example, it is picked up by another VM), it can be resumed from the last checkpoint.

Degrade gracefully. Sometimes you can't work around a problem, but you can provide reduced functionality that is still useful. Consider an application that shows a catalog of books. If the application can't retrieve the thumbnail image for the cover, it might show a placeholder image. Entire subsystems might be noncritical for the application. For example, in an e-commerce site, showing product recommendations is probably less critical than processing orders.

Throttle clients. Sometimes a small number of users create excessive load, which can reduce your application's availability for other users. In this situation, throttle the client for a certain period of time. See the [Throttling pattern](#).

Block bad actors. Just because you throttle a client, it doesn't mean client was acting maliciously. It just means the client exceeded their service quota. But if a client consistently exceeds their quota or otherwise behaves badly, you might block them. Define an out-of-band process for user to request getting unblocked.

Use leader election. When you need to coordinate a task, use [Leader Election](#) to select a coordinator. That way, the coordinator is not a single point of failure. If the coordinator fails, a new one is selected. Rather than implement a leader election algorithm from scratch, consider an off-the-shelf solution such as Zookeeper.

Test with fault injection. All too often, the success path is well tested but not the failure path. A system could run in production for a long time before a failure path is exercised. Use fault injection to test the resiliency of the system to failures, either by triggering actual failures or by simulating them.

Embrace chaos engineering. Chaos engineering extends the notion of fault injection, by randomly injecting failures or abnormal conditions into production instances.

For a structured approach to making your applications self healing, see [Design reliable applications for Azure](#).

Make all things redundant

3/10/2022 • 2 minutes to read • [Edit Online](#)

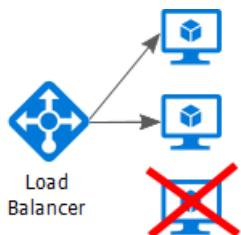
Build redundancy into your application, to avoid having single points of failure

A resilient application routes around failure. Identify the critical paths in your application. Is there redundancy at each point in the path? When a subsystem fails, will the application fail over to something else?

Recommendations

Consider business requirements. The amount of redundancy built into a system can affect both cost and complexity. Your architecture should be informed by your business requirements, such as recovery time objective (RTO). For example, a multi-region deployment is more expensive than a single-region deployment, and is more complicated to manage. You will need operational procedures to handle failover and fallback. The additional cost and complexity might be justified for some business scenarios and not others.

Place VMs behind a load balancer. Don't use a single VM for mission-critical workloads. Instead, place multiple VMs behind a load balancer. If any VM becomes unavailable, the load balancer distributes traffic to the remaining healthy VMs. To learn how to deploy this configuration, see [Multiple VMs for scalability and availability](#).

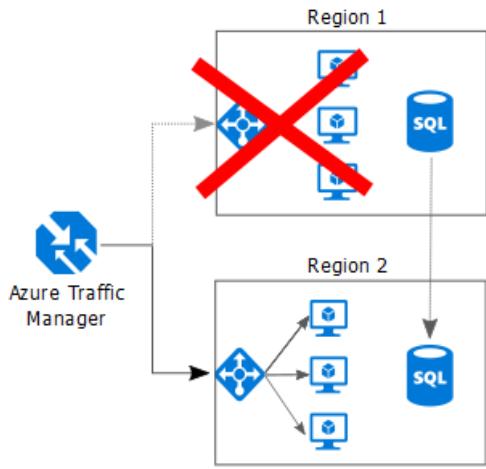


Replicate databases. Azure SQL Database and Cosmos DB automatically replicate the data within a region, and you can enable geo-replication across regions. If you are using an IaaS database solution, choose one that supports replication and failover, such as [SQL Server Always On availability groups](#).

Enable geo-replication. Geo-replication for [Azure SQL Database](#) and [Cosmos DB](#) creates secondary readable replicas of your data in one or more secondary regions. In the event of an outage, the database can fail over to the secondary region for writes.

Partition for availability. Database partitioning is often used to improve scalability, but it can also improve availability. If one shard goes down, the other shards can still be reached. A failure in one shard will only disrupt a subset of the total transactions.

Deploy to more than one region. For the highest availability, deploy the application to more than one region. That way, in the rare case when a problem affects an entire region, the application can fail over to another region. The following diagram shows a multi-region application that uses Azure Traffic Manager to handle failover.



Synchronize front and backend failover. Use Azure Traffic Manager to fail over the front end. If the front end becomes unreachable in one region, Traffic Manager will route new requests to the secondary region. Depending on your database solution, you may need to coordinate failing over the database.

Use automatic failover but manual fallback. Use Traffic Manager for automatic failover, but not for automatic fallback. Automatic fallback carries a risk that you might switch to the primary region before the region is completely healthy. Instead, verify that all application subsystems are healthy before manually failing back. Also, depending on the database, you might need to check data consistency before failing back.

Include redundancy for Traffic Manager. Traffic Manager is a possible failure point. Review the Traffic Manager SLA, and determine whether using Traffic Manager alone meets your business requirements for high availability. If not, consider adding another traffic management solution as a fallback. If the Azure Traffic Manager service fails, change your CNAME records in DNS to point to the other traffic management service.

Design to scale out

3/10/2022 • 2 minutes to read • [Edit Online](#)

Design your application so that it can scale horizontally

A primary advantage of the cloud is elastic scaling — the ability to use as much capacity as you need, scaling out as load increases, and scaling in when the extra capacity is not needed. Design your application so that it can scale horizontally, adding or removing new instances as demand requires.

Recommendations

Avoid instance stickiness. Stickiness, or *session affinity*, is when requests from the same client are always routed to the same server. Stickiness limits the application's ability to scale out. For example, traffic from a high-volume user will not be distributed across instances. Causes of stickiness include storing session state in memory, and using machine-specific keys for encryption. Make sure that any instance can handle any request.

Identify bottlenecks. Scaling out isn't a magic fix for every performance issue. For example, if your backend database is the bottleneck, it won't help to add more web servers. Identify and resolve the bottlenecks in the system first, before throwing more instances at the problem. Stateful parts of the system are the most likely cause of bottlenecks.

Decompose workloads by scalability requirements. Applications often consist of multiple workloads, with different requirements for scaling. For example, an application might have a public-facing site and a separate administration site. The public site may experience sudden surges in traffic, while the administration site has a smaller, more predictable load.

Offload resource-intensive tasks. Tasks that require a lot of CPU or I/O resources should be moved to [background jobs](#) when possible, to minimize the load on the front end that is handling user requests.

Use built-in autoscaling features. Many Azure compute services have built-in support for autoscaling. If the application has a predictable, regular workload, scale out on a schedule. For example, scale out during business hours. Otherwise, if the workload is not predictable, use performance metrics such as CPU or request queue length to trigger autoscaling. For autoscaling best practices, see [Autoscaling](#).

Consider aggressive autoscaling for critical workloads. For critical workloads, you want to keep ahead of demand. It's better to add new instances quickly under heavy load to handle the additional traffic, and then gradually scale back.

Design for scale in. Remember that with elastic scale, the application will have periods of scale in, when instances get removed. The application must gracefully handle instances being removed. Here are some ways to handle scalein:

- Listen for shutdown events (when available) and shut down cleanly.
- Clients/consumers of a service should support transient fault handling and retry.
- For long-running tasks, consider breaking up the work, using checkpoints or the [Pipes and Filters](#) pattern.
- Put work items on a queue so that another instance can pick up the work, if an instance is removed in the middle of processing.

Partition around limits

3/10/2022 • 2 minutes to read • [Edit Online](#)

Use partitioning to work around database, network, and compute limits

In the cloud, all services have limits in their ability to scale up. Azure service limits are documented in [Azure subscription and service limits, quotas, and constraints](#). Limits include number of cores, database size, query throughput, and network throughput. If your system grows sufficiently large, you may hit one or more of these limits. Use partitioning to work around these limits.

There are many ways to partition a system, such as:

- Partition a database to avoid limits on database size, data I/O, or number of concurrent sessions.
- Partition a queue or message bus to avoid limits on the number of requests or the number of concurrent connections.
- Partition an App Service web app to avoid limits on the number of instances per App Service plan.

A database can be partitioned *horizontally*, *vertically*, or *functionally*.

- In horizontal partitioning, also called sharding, each partition holds data for a subset of the total data set. The partitions share the same data schema. For example, customers whose names start with A–M go into one partition, N–Z into another partition.
- In vertical partitioning, each partition holds a subset of the fields for the items in the data store. For example, put frequently accessed fields in one partition, and less frequently accessed fields in another.
- In functional partitioning, data is partitioned according to how it is used by each bounded context in the system. For example, store invoice data in one partition and product inventory data in another. The schemas are independent.

For more detailed guidance, see [Data partitioning](#).

Recommendations

Partition different parts of the application. Databases are one obvious candidate for partitioning, but also consider storage, cache, queues, and compute instances.

Design the partition key to avoid hotspots. If you partition a database, but one shard still gets the majority of the requests, then you haven't solved your problem. Ideally, load gets distributed evenly across all the partitions. For example, hash by customer ID and not the first letter of the customer name, because some letters are more frequent. The same principle applies when partitioning a message queue. Pick a partition key that leads to an even distribution of messages across the set of queues. For more information, see [Sharding](#).

Partition around Azure subscription and service limits. Individual components and services have limits, but there are also limits for subscriptions and resource groups. For very large applications, you might need to partition around those limits.

Partition at different levels. Consider a database server deployed on a VM. The VM has a VHD that is backed by Azure Storage. The storage account belongs to an Azure subscription. Notice that each step in the hierarchy has limits. The database server may have a connection pool limit. VMs have CPU and network limits. Storage has IOPS limits. The subscription has limits on the number of VM cores. Generally, it's easier to partition lower in the

hierarchy. Only large applications should need to partition at the subscription level.

Design for operations

3/10/2022 • 2 minutes to read • [Edit Online](#)

Design an application so that the operations team has the tools they need

The cloud has dramatically changed the role of the operations team. They are no longer responsible for managing the hardware and infrastructure that hosts the application. That said, operations is still a critical part of running a successful cloud application. Some of the important functions of the operations team include:

- Deployment
- Monitoring
- Escalation
- Incident response
- Security auditing

Robust logging and tracing are particularly important in cloud applications. Involve the operations team in design and planning, to ensure the application gives them the data and insight they need to be successful.

Recommendations

Make all things observable. Once a solution is deployed and running, logs and traces are your primary insight into the system. *Tracing* records a path through the system, and is useful to pinpoint bottlenecks, performance issues, and failure points. *Logging* captures individual events such as application state changes, errors, and exceptions. Log in production, or else you lose insight at the very times when you need it the most.

Instrument for monitoring. Monitoring gives insight into how well (or poorly) an application is performing, in terms of availability, performance, and system health. For example, monitoring tells you whether you are meeting your SLA. Monitoring happens during the normal operation of the system. It should be as close to real-time as possible, so that the operations staff can react to issues quickly. Ideally, monitoring can help avert problems before they lead to a critical failure. For more information, see [Monitoring and diagnostics](#).

Instrument for root cause analysis. Root cause analysis is the process of finding the underlying cause of failures. It occurs after a failure has already happened.

Use distributed tracing. Use a distributed tracing system that is designed for concurrency, asynchrony, and cloud scale. Traces should include a correlation ID that flows across service boundaries. A single operation may involve calls to multiple application services. If an operation fails, the correlation ID helps to pinpoint the cause of the failure.

Standardize logs and metrics. The operations team will need to aggregate logs from across the various services in your solution. If every service uses its own logging format, it becomes difficult or impossible to get useful information from them. Define a common schema that includes fields such as correlation ID, event name, IP address of the sender, and so forth. Individual services can derive custom schemas that inherit the base schema, and contain additional fields.

Automate management tasks, including provisioning, deployment, and monitoring. Automating a task makes it repeatable and less prone to human errors.

Treat configuration as code. Check configuration files into a version control system, so that you can track and version your changes, and roll back if needed.

Use platform as a service (PaaS) options

3/10/2022 • 2 minutes to read • [Edit Online](#)

When possible, use platform as a service (PaaS) rather than infrastructure as a service (IaaS)

IaaS is like having a box of parts. You can build anything, but you have to assemble it yourself. PaaS options are easier to configure and administer. You don't need to provision VMs, set up VNets, manage patches and updates, and all of the other overhead associated with running software on a VM.

For example, suppose your application needs a message queue. You could set up your own messaging service on a VM, using something like RabbitMQ. But Azure Service Bus already provides reliable messaging as service, and it's simpler to set up. Just create a Service Bus namespace (which can be done as part of a deployment script) and then call Service Bus using the client SDK.

Of course, your application may have specific requirements that make an IaaS approach more suitable. However, even if your application is based on IaaS, look for places where it may be natural to incorporate PaaS options. These include cache, queues, and data storage.

INSTEAD OF RUNNING...	CONSIDER USING...
Active Directory	Azure Active Directory
Elasticsearch	Azure Search
Hadoop	HDInsight
IIS	App Service
MongoDB	Cosmos DB
Redis	Azure Cache for Redis
SQL Server	Azure SQL Database
File share	Azure NetApp Files

Please note that this is not meant to be an exhaustive list, but a subset of equivalent options.

Use the best data store for the job

3/10/2022 • 2 minutes to read • [Edit Online](#)

Pick the storage technology that is the best fit for your data and how it will be used

Gone are the days when you would just stick all of your data into a big relational SQL database. Relational databases are very good at what they do — providing ACID guarantees for transactions over relational data. But they come with some costs:

- Queries may require expensive joins.
- Data must be normalized and conform to a predefined schema (schema on write).
- Lock contention may impact performance.

In any large solution, it's likely that a single data store technology won't fill all your needs. Alternatives to relational databases include key/value stores, document databases, search engine databases, time series databases, column family databases, and graph databases. Each has pros and cons, and different types of data fit more naturally into one or another.

For example, you might store a product catalog in a document database, such as Cosmos DB, which allows for a flexible schema. In that case, each product description is a self-contained document. For queries over the entire catalog, you might index the catalog and store the index in Azure Search. Product inventory might go into a SQL database, because that data requires ACID guarantees.

Remember that data includes more than just the persisted application data. It also includes application logs, events, messages, and caches.

Recommendations

Don't use a relational database for everything. Consider other data stores when appropriate. See [Choose the right data store](#).

Embrace polyglot persistence. In any large solution, it's likely that a single data store technology won't fill all your needs.

Consider the type of data. For example, put transactional data into SQL, put JSON documents into a document database, put telemetry data into a time series data base, put application logs in Elasticsearch, and put blobs in Azure Blob Storage.

Prefer availability over (strong) consistency. The CAP theorem implies that a distributed system must make trade-offs between availability and consistency. (Network partitions, the other leg of the CAP theorem, can never be completely avoided.) Often, you can achieve higher availability by adopting an *eventual consistency* model.

Consider the skillset of the development team. There are advantages to using polyglot persistence, but it's possible to go overboard. Adopting a new data storage technology requires a new set of skills. The development team must understand how to get the most out of the technology. They must understand appropriate usage patterns, how to optimize queries, tune for performance, and so on. Factor this in when considering storage technologies.

Use compensating transactions. A side effect of polyglot persistence is that single transaction might write data to multiple stores. If something fails, use compensating transactions to undo any steps that already

completed.

Look at bounded contexts. *Bounded context* is a term from domain driven design. A bounded context is an explicit boundary around a domain model, and defines which parts of the domain the model applies to. Ideally, a bounded context maps to a subdomain of the business domain. The bounded contexts in your system are a natural place to consider polyglot persistence. For example, "products" may appear in both the Product Catalog subdomain and the Product Inventory subdomain, but it's very likely that these two subdomains have different requirements for storing, updating, and querying products.

Design for evolution

3/10/2022 • 3 minutes to read • [Edit Online](#)

An evolutionary design is key for continuous innovation

All successful applications change over time, whether to fix bugs, add new features, bring in new technologies, or make existing systems more scalable and resilient. If all the parts of an application are tightly coupled, it becomes very hard to introduce changes into the system. A change in one part of the application may break another part, or cause changes to ripple through the entire codebase.

This problem is not limited to monolithic applications. An application can be decomposed into services, but still exhibit the sort of tight coupling that leaves the system rigid and brittle. But when services are designed to evolve, teams can innovate and continuously deliver new features.

Microservices are becoming a popular way to achieve an evolutionary design, because they address many of the considerations listed here.

Recommendations

Enforce high cohesion and loose coupling. A service is *cohesive* if it provides functionality that logically belongs together. Services are *loosely coupled* if you can change one service without changing the other. High cohesion generally means that changes in one function will require changes in other related functions. If you find that updating a service requires coordinated updates to other services, it may be a sign that your services are not cohesive. One of the goals of domain-driven design (DDD) is to identify those boundaries.

Encapsulate domain knowledge. When a client consumes a service, the responsibility for enforcing the business rules of the domain should not fall on the client. Instead, the service should encapsulate all of the domain knowledge that falls under its responsibility. Otherwise, every client has to enforce the business rules, and you end up with domain knowledge spread across different parts of the application.

Use asynchronous messaging. Asynchronous messaging is a way to decouple the message producer from the consumer. The producer does not depend on the consumer responding to the message or taking any particular action. With a pub/sub architecture, the producer may not even know who is consuming the message. New services can easily consume the messages without any modifications to the producer.

Don't build domain knowledge into a gateway. Gateways can be useful in a microservices architecture, for things like request routing, protocol translation, load balancing, or authentication. However, the gateway should be restricted to this sort of infrastructure functionality. It should not implement any domain knowledge, to avoid becoming a heavy dependency.

Expose open interfaces. Avoid creating custom translation layers that sit between services. Instead, a service should expose an API with a well-defined API contract. The API should be versioned, so that you can evolve the API while maintaining backward compatibility. That way, you can update a service without coordinating updates to all of the upstream services that depend on it. Public facing services should expose a RESTful API over HTTP. Backend services might use an RPC-style messaging protocol for performance reasons.

Design and test against service contracts. When services expose well-defined APIs, you can develop and test against those APIs. That way, you can develop and test an individual service without spinning up all of its dependent services. (Of course, you would still perform integration and load testing against the real services.)

Abstract infrastructure away from domain logic. Don't let domain logic get mixed up with infrastructure-related functionality, such as messaging or persistence. Otherwise, changes in the domain logic will require

updates to the infrastructure layers and vice versa.

Offload cross-cutting concerns to a separate service. For example, if several services need to authenticate requests, you could move this functionality into its own service. Then you could evolve the authentication service — for example, by adding a new authentication flow — without touching any of the services that use it.

Deploy services independently. When the DevOps team can deploy a single service independently of other services in the application, updates can happen more quickly and safely. Bug fixes and new features can be rolled out at a more regular cadence. Design both the application and the release process to support independent updates.

Build for the needs of the business

3/10/2022 • 2 minutes to read • [Edit Online](#)

Every design decision must be justified by a business requirement

This design principle may seem obvious, but it's crucial to keep in mind when designing a solution. Do you anticipate millions of users, or a few thousand? Is a one-hour application outage acceptable? Do you expect large bursts in traffic or a predictable workload? Ultimately, every design decision must be justified by a business requirement.

Recommendations

Define business objectives, including the recovery time objective (RTO), recovery point objective (RPO), and maximum tolerable outage (MTO). These numbers should inform decisions about the architecture. For example, to achieve a low RTO, you might implement automated failover to a secondary region. But if your solution can tolerate a higher RTO, that degree of redundancy might be unnecessary.

Document service level agreements (SLA) and service level objectives (SLO), including availability and performance metrics. You might build a solution that delivers 99.95% availability. Is that enough? The answer is a business decision.

Model the application around the business domain. Start by analyzing the business requirements. Use these requirements to model the application. Consider using a domain-driven design (DDD) approach to create **domain models** that reflect the business processes and use cases.

Capture both functional and nonfunctional requirements. Functional requirements let you judge whether the application does the right thing. Nonfunctional requirements let you judge whether the application does those things *well*. In particular, make sure that you understand your requirements for scalability, availability, and latency. These requirements will influence design decisions and choice of technology.

Decompose by workload. The term "workload" in this context means a discrete capability or computing task, which can be logically separated from other tasks. Different workloads may have different requirements for availability, scalability, data consistency, and disaster recovery.

Plan for growth. A solution might meet your current needs, in terms of number of users, volume of transactions, data storage, and so forth. However, a robust application can handle growth without major architectural changes. See [Design to scale out](#) and [Partition around limits](#). Also consider that your business model and business requirements will likely change over time. If an application's service model and data models are too rigid, it becomes hard to evolve the application for new use cases and scenarios. See [Design for evolution](#).

Manage costs. In a traditional on-premises application, you pay upfront for hardware as a capital expenditure. In a cloud application, you pay for the resources that you consume. Make sure that you understand the pricing model for the services that you consume. The total cost will include network bandwidth usage, storage, IP addresses, service consumption, and other factors. For more information, see [Azure pricing](#). Also consider your operations costs. In the cloud, you don't have to manage the hardware or other infrastructure, but you still need to manage your applications, including DevOps, incident response, disaster recovery, and so forth.

Choose a Kubernetes at the edge compute option

3/10/2022 • 6 minutes to read • [Edit Online](#)

This document discusses the trade-offs for various options available for extending compute on the edge. The following considerations for each Kubernetes option are covered:

- **Operational cost.** The expected labor required to maintain and operate the Kubernetes clusters.
- **Ease of configuration.** The level of difficulty to configure and deploy a Kubernetes cluster.
- **Flexibility.** A measure of how adaptable the Kubernetes option is to integrate a customized configuration with existing infrastructure at the edge.
- **Mixed node.** Ability to run a Kubernetes cluster with both Linux and Windows nodes.

Assumptions

- You are a cluster operator looking to understand different options for running Kubernetes at the edge and managing clusters in Azure.
- You have a good understanding of existing infrastructure and any other infrastructure requirements, including storage and networking requirements.

After reading this document, you'll be in a better position to identify which option best fits your scenario and the environment required.

Kubernetes choices at a glance

	OPERATIONAL COST	EASE OF CONFIGURATION	FLEXIBILITY	MIXED NODE	SUMMARY
Bare-metal Kubernetes	High**	Difficult**	High**	Yes	A ground-up configuration on any available infrastructure at location with the option to use Azure Arc for added Azure capabilities.
K8s on Azure Stack Edge Pro	Low	Easy	Low	Linux only	Kubernetes deployed on Azure Stack Edge appliance deployed at location.
AKS on HCI	Low	Easy	Medium	Yes	AKS deployed on Azure Stack HCI or Windows Server 2019.

*Other managed edge platforms (OpenShift, Tanzu, and so on) aren't in scope for this document.

**These values are based on using `kubeadm`, for the sake of simplicity. Different options for running bare-metal Kubernetes at the edge would alter the rating in these categories.

Bare-metal Kubernetes

Ground-up configuration of Kubernetes using tools like `kubeadm` on any underlying infrastructure.

The biggest constraints for bare-metal Kubernetes are around the specific needs and requirements of the organization. The opportunity to use any distribution, networking interface, and plugin means higher complexity and operational cost. But this offers the most flexible option for customizing your cluster.

Scenario

Often, *edge* locations have specific requirements for running Kubernetes clusters that aren't met with the other Azure solutions described in this document. Meaning this option is typically best for those unable to use managed services due to unsupported existing infrastructure, or those who seek to have maximum control of their clusters.

- This option can be especially difficult for those who are new to Kubernetes. This isn't uncommon for organizations looking to run edge clusters. Options like [MicroK8s](#) or [k3s](#) aim to flatten that learning curve.
- It's important to understand any underlying infrastructure and any integration that is expected to take place up front. This will help to narrow down viable options and to identify any gaps with the open-source tooling and/or plugins.
- Enabling clusters with [Azure Arc](#) presents a simple way to manage your cluster from Azure alongside other resources. This also brings other Azure capabilities to your cluster, including [Azure Policy](#), [Azure Monitor](#), [Microsoft Defender for Cloud](#), and other services.
- Because cluster configuration isn't trivial, it's especially important to be mindful of CI/CD. Tracking and acting on upstream changes of various plugins, and making sure those changes don't affect the health of your cluster, becomes a direct responsibility. It's important for you to have a strong CI/CD solution, strong testing, and monitoring in place.

Tooling options

Cluster bootstrap:

- [kubeadm](#): Kubernetes tool for creating ground-up Kubernetes clusters. Good for standard compute resources (Linux/Windows).
- [MicroK8s](#): Simplified administration and configuration ("LowOps"), conformant Kubernetes by Canonical.
- [k3s](#): Certified Kubernetes distribution built for Internet of Things (IoT) and edge computing.

Storage:

- Explore available [CSI drivers](#): Many options are available to fit your requirements from cloud to local file shares.

Networking:

- A full list of available add-ons can be found here: [Networking add-ons](#). Some popular options include [Flannel](#), a simple overlay network, and [Calico](#), which provides a full networking stack.

Considerations

Operational cost:

- Without the support that comes with managed services, it's up to the organization to maintain and operate

the cluster as a whole (storage, networking, upgrades, observability, application management). The operational cost is considered high.

Ease of configuration:

- Evaluating the many open-source options at every stage of configuration whether its networking, storage, or monitoring options is inevitable and can become complex. Requires more consideration for configuring a CI/CD for cluster configuration. Because of these concerns, the ease of configuration is considered difficult.

Flexibility:

- With the ability to use any open-source tool or plugin without any provider restrictions, bare-metal Kubernetes is highly flexible.

Kubernetes on Azure Stack Edge

Kubernetes cluster (a master VM and a worker VM) configured and deployed for you on your Azure Stack Edge Pro device.

[Azure Stack Edge Pro](#) devices deliver Azure capabilities like compute, storage, networking, and hardware-accelerated machine learning (ML) to any edge location. Kubernetes clusters can be created once the compute role is enabled on any of the Pro-GPU, Pro-R, and Mini-R devices. Managing upgrades of the Kubernetes cluster can be done using standard updates available for the device.

Scenario

Ideal for those with existing (Linux) IoT workloads or upgrading their compute for ML at the edge. This is a good option when it isn't necessary to have more granular control over the clusters.

- Admin permissions aren't granted by default. Although you can work with the product group to make certain exceptions, this makes it difficult to have finer control of your cluster.
- There is an extra [cost](#) if there isn't already an Azure Stack Edge device. Explore [Azure Stack Edge devices](#) and see if any fit your compute requirements.
- [Calico](#), [MetallLB](#), and [CoreDNS](#) are installed for Kubernetes networking on the device.
- Only [Linux](#) workloads are supported at this time.
- In addition to Kubernetes, Azure Stack Edge also comes with the IoT runtime, which means that workloads may also be deployed to your Azure Stack Edge clusters via IoT Edge.
- Support for two node clusters isn't currently available. This effectively means that this option is *not* a highly available (HA) solution.

Considerations

Operational cost:

- With the support that comes with the device, operational cost is minimal and is scoped to workload management.

Ease of configuration:

- Pre-configured and well-documented Kubernetes cluster deployment simplifies the configuration required compared to bare-metal Kubernetes.

Flexibility:

- Configuration is already set, and Admin permissions aren't granted by default. Product group involvement may be required beyond basic configuration, and the underlying infrastructure must be an Azure Stack Edge

Pro device, making this a less flexible option.

AKS on HCI

Note: This option is currently in [preview](#).

AKS-HCI is a set of predefined settings and configurations that is used to deploy one or more Kubernetes clusters (with Windows Admin Center or PowerShell modules) on a multi-node cluster running either Windows Server 2019 Datacenter or Azure Stack HCI 20H2.

Scenario

Ideal for those who want a simplified and streamlined way to get a Microsoft-supported cluster on compatible devices (Azure Stack HCI or Windows Server 2019 Datacenter). Operations and configuration complexity are reduced at the expense of the flexibility when compared to the bare-metal Kubernetes option.

Considerations

At the time of this writing, the preview comes with many limitations (permissions, networking limitations, large compute requirements, and documentation gaps). Purposes other than evaluation and development are discouraged at this time.

Operational cost:

- Microsoft-supported cluster minimizes operational costs.

Ease of configuration:

- Pre-configured and well-documented Kubernetes cluster deployment simplifies the configuration required compared to bare-metal Kubernetes.

Flexibility:

- Cluster configuration itself is set, but Admin permissions are granted. The underlying infrastructure must either be Azure Stack HCI or Windows Server 2019. This option is more flexible than Kubernetes on Azure Stack Edge and less flexible than bare-metal Kubernetes.

Next steps

For more information, see the following articles:

- [What is Azure IoT Edge](#)
- [Kubernetes on your Azure Stack Edge Pro GPU device](#)
- [Use IoT Edge module to run a Kubernetes stateless application on your Azure Stack Edge Pro GPU device](#)
- [Deploy a Kubernetes stateless application via kubectl on your Azure Stack Edge Pro GPU device](#)
- [AI at the edge with Azure Stack Hub](#)
- [Building a CI/CD pipeline for microservices on Kubernetes](#)
- [Use Kubernetes dashboard to monitor your Azure Stack Edge Pro GPU device](#)

Understand data store models

3/10/2022 • 12 minutes to read • [Edit Online](#)

Modern business systems manage increasingly large volumes of heterogeneous data. This heterogeneity means that a single data store is usually not the best approach. Instead, it's often better to store different types of data in different data stores, each focused toward a specific workload or usage pattern. The term *polyglot persistence* is used to describe solutions that use a mix of data store technologies. Therefore, it's important to understand the main storage models and their tradeoffs.

Selecting the right data store for your requirements is a key design decision. There are literally hundreds of implementations to choose from among SQL and NoSQL databases. Data stores are often categorized by how they structure data and the types of operations they support. This article describes several of the most common storage models. Note that a particular data store technology may support multiple storage models. For example, a relational database management systems (RDBMS) may also support key/value or graph storage. In fact, there is a general trend for so-called *multi-model* support, where a single database system supports several models. But it's still useful to understand the different models at a high level.

Not all data stores in a given category provide the same feature-set. Most data stores provide server-side functionality to query and process data. Sometimes this functionality is built into the data storage engine. In other cases, the data storage and processing capabilities are separated, and there may be several options for processing and analysis. Data stores also support different programmatic and management interfaces.

Generally, you should start by considering which storage model is best suited for your requirements. Then consider a particular data store within that category, based on factors such as feature set, cost, and ease of management.

NOTE

Learn more about identifying and reviewing your data service requirements for cloud adoption, in the [Microsoft Cloud Adoption Framework for Azure](#). Likewise, you can also learn about [selecting storage tools and services](#).

Relational database management systems

Relational databases organize data as a series of two-dimensional tables with rows and columns. Most vendors provide a dialect of the Structured Query Language (SQL) for retrieving and managing data. An RDBMS typically implements a transactionally consistent mechanism that conforms to the ACID (Atomic, Consistent, Isolated, Durable) model for updating information.

An RDBMS typically supports a schema-on-write model, where the data structure is defined ahead of time, and all read or write operations must use the schema.

This model is very useful when strong consistency guarantees are important — where all changes are atomic, and transactions always leave the data in a consistent state. However, an RDBMS generally can't scale out horizontally without sharding the data in some way. Also, the data in an RDBMS must be normalized, which isn't appropriate for every data set.

Azure services

- [Azure SQL Database | \(Security Baseline\)](#)
- [Azure Database for MySQL | \(Security Baseline\)](#)
- [Azure Database for PostgreSQL | \(Security Baseline\)](#)

- [Azure Database for MariaDB | \(Security Baseline\)](#)

Workload

- Records are frequently created and updated.
- Multiple operations have to be completed in a single transaction.
- Relationships are enforced using database constraints.
- Indexes are used to optimize query performance.

Data type

- Data is highly normalized.
- Database schemas are required and enforced.
- Many-to-many relationships between data entities in the database.
- Constraints are defined in the schema and imposed on any data in the database.
- Data requires high integrity. Indexes and relationships need to be maintained accurately.
- Data requires strong consistency. Transactions operate in a way that ensures all data are 100% consistent for all users and processes.
- Size of individual data entries is small to medium-sized.

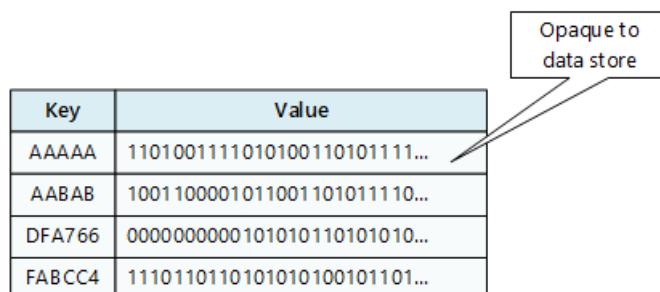
Examples

- Inventory management
- Order management
- Reporting database
- Accounting

Key/value stores

A key/value store associates each data value with a unique key. Most key/value stores only support simple query, insert, and delete operations. To modify a value (either partially or completely), an application must overwrite the existing data for the entire value. In most implementations, reading or writing a single value is an atomic operation.

An application can store arbitrary data as a set of values. Any schema information must be provided by the application. The key/value store simply retrieves or stores the value by key.



Key	Value
AAAAAA	11010011101010011010111...
AABAB	1001100001011001101011110...
DFA766	0000000000101010110101010...
FABCC4	11101101101010100101101...

Key/value stores are highly optimized for applications performing simple lookups, but are less suitable if you need to query data across different key/value stores. Key/value stores are also not optimized for querying by value.

A single key/value store can be extremely scalable, as the data store can easily distribute data across multiple nodes on separate machines.

Azure services

- [Azure Cosmos DB Table API and SQL API | \(Cosmos DB Security Baseline\)](#)
- [Azure Cache for Redis | \(Security Baseline\)](#)
- [Azure Table Storage | \(Security Baseline\)](#)

Workload

- Data is accessed using a single key, like a dictionary.
- No joins, lock, or unions are required.
- No aggregation mechanisms are used.
- Secondary indexes are generally not used.

Data type

- Each key is associated with a single value.
- There is no schema enforcement.
- No relationships between entities.

Examples

- Data caching
- Session management
- User preference and profile management
- Product recommendation and ad serving

Document databases

A document database stores a collection of *documents*, where each document consists of named fields and data. The data can be simple values or complex elements such as lists and child collections. Documents are retrieved by unique keys.

Typically, a document contains the data for single entity, such as a customer or an order. A document may contain information that would be spread across several relational tables in an RDBMS. Documents don't need to have the same structure. Applications can store different data in documents as business requirements change.

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Azure service

- [Azure Cosmos DB SQL API | \(Cosmos DB Security Baseline\)](#)

Workload

- Insert and update operations are common.
- No object-relational impedance mismatch. Documents can better match the object structures used in application code.
- Individual documents are retrieved and written as a single block.

- Data requires index on multiple fields.

Data type

- Data can be managed in de-normalized way.
- Size of individual document data is relatively small.
- Each document type can use its own schema.
- Documents can include optional fields.
- Document data is semi-structured, meaning that data types of each field are not strictly defined.

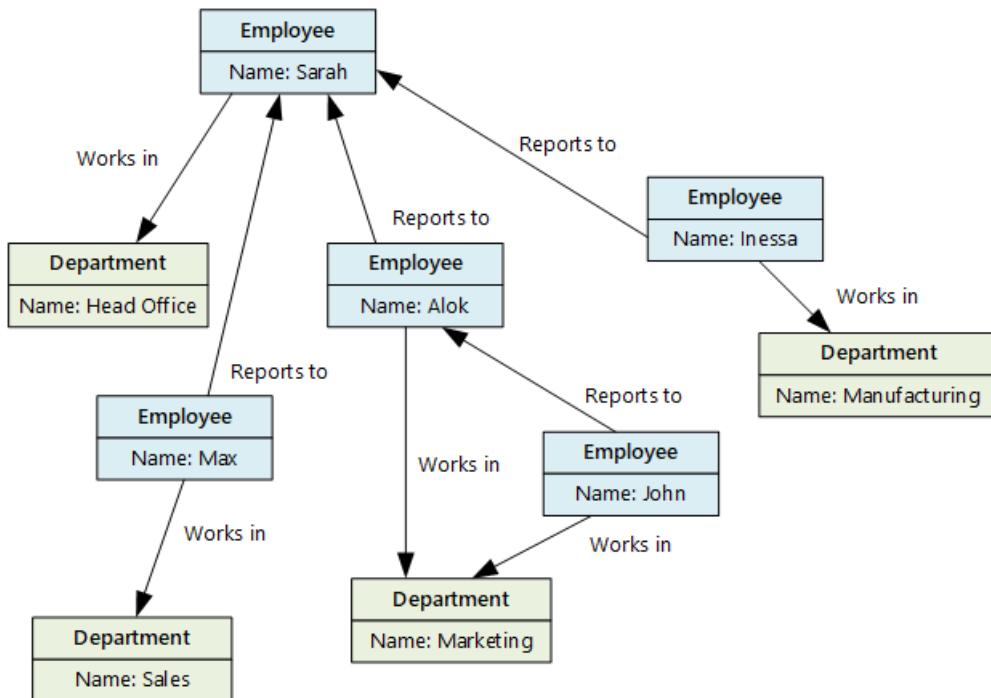
Examples

- Product catalog
- Content management
- Inventory management

Graph databases

A graph database stores two types of information, nodes and edges. Edges specify relationships between nodes. Nodes and edges can have properties that provide information about that node or edge, similar to columns in a table. Edges can also have a direction indicating the nature of the relationship.

Graph databases can efficiently perform queries across the network of nodes and edges and analyze the relationships between entities. The following diagram shows an organization's personnel database structured as a graph. The entities are employees and departments, and the edges indicate reporting relationships and the departments in which employees work.



This structure makes it straightforward to perform queries such as "Find all employees who report directly or indirectly to Sarah" or "Who works in the same department as John?" For large graphs with lots of entities and relationships, you can perform very complex analyses very quickly. Many graph databases provide a query language that you can use to traverse a network of relationships efficiently.

Azure services

- [Azure Cosmos DB Gremlin API | \(Security Baseline\)](#)
- [SQL Server | \(Security Baseline\)](#)

Workload

- Complex relationships between data items involving many hops between related data items.
- The relationship between data items are dynamic and change over time.
- Relationships between objects are first-class citizens, without requiring foreign-keys and joins to traverse.

Data type

- Nodes and relationships.
- Nodes are similar to table rows or JSON documents.
- Relationships are just as important as nodes, and are exposed directly in the query language.
- Composite objects, such as a person with multiple phone numbers, tend to be broken into separate, smaller nodes, combined with traversable relationships

Examples

- Organization charts
- Social graphs
- Fraud detection
- Recommendation engines

Data analytics

Data analytics stores provide massively parallel solutions for ingesting, storing, and analyzing data. The data is distributed across multiple servers to maximize scalability. Large data file formats such as delimiter files (CSV), [parquet](#), and [ORC](#) are widely used in data analytics. Historical data is typically stored in data stores such as blob storage or [Azure Data Lake Storage Gen2](#), which are then accessed by Azure Synapse, Databricks, or HDInsight as external tables. A typical scenario using data stored as parquet files for performance, is described in the article [Use external tables with Synapse SQL](#).

Azure services

- [Azure Synapse Analytics | \(Security Baseline\)](#)
- [Azure Data Lake | \(Security Baseline\)](#)
- [Azure Data Explorer | \(Security Baseline\)](#)
- [Azure Analysis Services](#)
- [HDInsight | \(Security Baseline\)](#)
- [Azure Databricks | \(Security Baseline\)](#)

Workload

- Data analytics
- Enterprise BI

Data type

- Historical data from multiple sources.
- Usually denormalized in a "star" or "snowflake" schema, consisting of fact and dimension tables.
- Usually loaded with new data on a scheduled basis.
- Dimension tables often include multiple historic versions of an entity, referred to as a *slowly changing dimension*.

Examples

- Enterprise data warehouse

Column-family databases

A column-family database organizes data into rows and columns. In its simplest form, a column-family database can appear very similar to a relational database, at least conceptually. The real power of a column-family

database lies in its denormalized approach to structuring sparse data.

You can think of a column-family database as holding tabular data with rows and columns, but the columns are divided into groups known as *column families*. Each column family holds a set of columns that are logically related together and are typically retrieved or manipulated as a unit. Other data that is accessed separately can be stored in separate column families. Within a column family, new columns can be added dynamically, and rows can be sparse (that is, a row doesn't need to have a value for every column).

The following diagram shows an example with two column families, `Identity` and `Contact Info`. The data for a single entity has the same row key in each column-family. This structure, where the rows for any given object in a column family can vary dynamically, is an important benefit of the column-family approach, making this form of data store highly suited for storing structured, volatile data.

CustomerID	Column Family: Identity	CustomerID	Column Family: Contact Info
001	First name: Mu Bae Last name: Min	001	Phone number: 555-0100 Email: someone@example.com
002	First name: Francisco Last name: Vila Nova Suffix: Jr.	002	Email: vilanova@contoso.com
003	First name: Lena Last name: Adamczyz Title: Dr.	003	Phone number: 555-0120

Unlike a key/value store or a document database, most column-family databases store data in key order, rather than by computing a hash. Many implementations allow you to create indexes over specific columns in a column-family. Indexes let you retrieve data by columns value, rather than row key.

Read and write operations for a row are usually atomic with a single column-family, although some implementations provide atomicity across the entire row, spanning multiple column-families.

Azure services

- [Azure Cosmos DB Cassandra API | \(Security Baseline\)](#)
- [HBase in HDInsight | \(Security Baseline\)](#)

Workload

- Most column-family databases perform write operations extremely quickly.
- Update and delete operations are rare.
- Designed to provide high throughput and low-latency access.
- Supports easy query access to a particular set of fields within a much larger record.
- Massively scalable.

Data type

- Data is stored in tables consisting of a key column and one or more column families.
- Specific columns can vary by individual rows.
- Individual cells are accessed via get and put commands
- Multiple rows are returned using a scan command.

Examples

- Recommendations
- Personalization
- Sensor data
- Telemetry
- Messaging
- Social media analytics

- Web analytics
- Activity monitoring
- Weather and other time-series data

Search Engine Databases

A search engine database allows applications to search for information held in external data stores. A search engine database can index massive volumes of data and provide near real-time access to these indexes.

Indexes can be multi-dimensional and may support free-text searches across large volumes of text data.

Indexing can be performed using a pull model, triggered by the search engine database, or using a push model, initiated by external application code.

Searching can be exact or fuzzy. A fuzzy search finds documents that match a set of terms and calculates how closely they match. Some search engines also support linguistic analysis that can return matches based on synonyms, genre expansions (for example, matching `dogs` to `pets`), and stemming (matching words with the same root).

Azure service

- [Azure Search | \(Security Baseline\)](#)

Workload

- Data indexes from multiple sources and services.
- Queries are ad-hoc and can be complex.
- Full text search is required.
- Ad hoc self-service query is required.

Data type

- Semi-structured or unstructured text
- Text with reference to structured data

Examples

- Product catalogs
- Site search
- Logging

Time series databases

Time series data is a set of values organized by time. Time series databases typically collect large amounts of data in real time from a large number of sources. Updates are rare, and deletes are often done as bulk operations. Although the records written to a time-series database are generally small, there are often a large number of records, and total data size can grow rapidly.

Azure service

- [Azure Time Series Insights](#)

Workload

- Records are generally appended sequentially in time order.
- An overwhelming proportion of operations (95-99%) are writes.
- Updates are rare.
- Deletes occur in bulk, and are made to contiguous blocks or records.
- Data is read sequentially in either ascending or descending time order, often in parallel.

Data type

- A timestamp is used as the primary key and sorting mechanism.
- Tags may define additional information about the type, origin, and other information about the entry.

Examples

- Monitoring and event telemetry.
- Sensor or other IoT data.

Object storage

Object storage is optimized for storing and retrieving large binary objects (images, files, video and audio streams, large application data objects and documents, virtual machine disk images). Large data files are also popularly used in this model, for example, delimiter file (CSV), [parquet](#), and [ORC](#). Object stores can manage extremely large amounts of unstructured data.

Azure service

- [Azure Blob Storage | \(Security Baseline\)](#)
- [Azure Data Lake Storage Gen2 | \(Security Baseline\)](#)

Workload

- Identified by key.
- Content is typically an asset such as a delimiter, image, or video file.
- Content must be durable and external to any application tier.

Data type

- Data size is large.
- Value is opaque.

Examples

- Images, videos, office documents, PDFs
- Static HTML, JSON, CSS
- Log and audit files
- Database backups

Shared files

Sometimes, using simple flat files can be the most effective means of storing and retrieving information. Using file shares enables files to be accessed across a network. Given appropriate security and concurrent access control mechanisms, sharing data in this way can enable distributed services to provide highly scalable data access for performing basic, low-level operations such as simple read and write requests.

Azure service

- [Azure Files | \(Security Baseline\)](#)

Workload

- Migration from existing apps that interact with the file system.
- Requires SMB interface.

Data type

- Files in a hierarchical set of folders.
- Accessible with standard I/O libraries.

Examples

- Legacy files

- Shared content accessible among a number of VMs or app instances

Aided with this understanding of different data storage models, the next step is to evaluate your workload and application, and decide which data store will meet your specific needs. Use the [data storage decision tree](#) to help with this process.

Select an Azure data store for your application

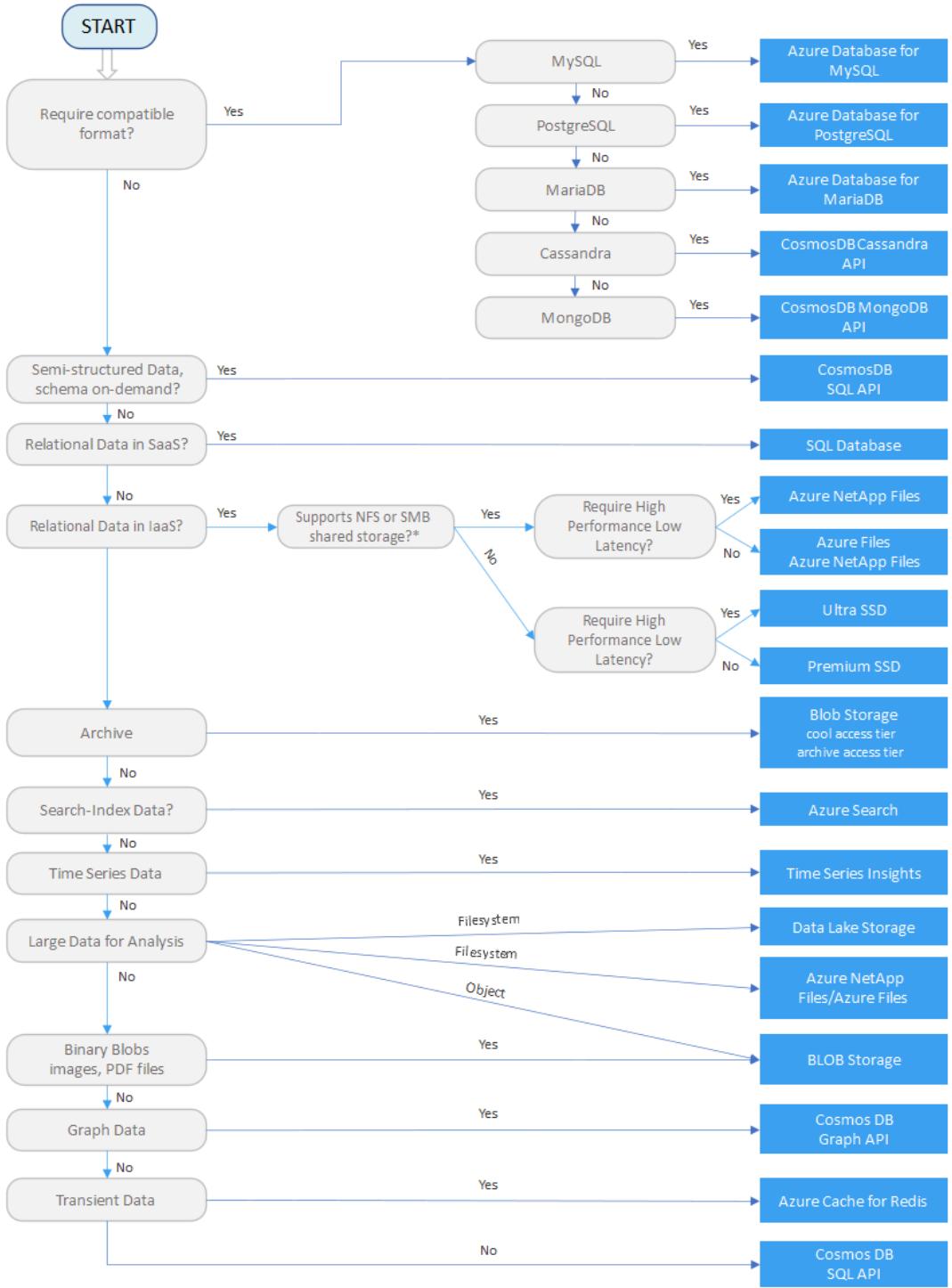
3/10/2022 • 2 minutes to read • [Edit Online](#)

Azure offers a number of managed data storage solutions, each providing different features and capabilities. This article will help you to choose a managed data store for your application.

If your application consists of multiple workloads, evaluate each workload separately. A complete solution may incorporate multiple data stores.

Select a candidate

Use the following flowchart to select a candidate Azure managed data store.



The output from this flowchart is a **starting point** for consideration. Next, perform a more detailed evaluation of the data store to see if it meets your needs. Refer to [Criteria for choosing a data store](#) to aid in this evaluation.

Choose specialized storage

Alternative database solutions often require specific storage solutions. For example, SAP HANA on VMs often employs Azure NetApp Files as its underlying storage solution. Evaluate your vendor's requirements to find an appropriate storage solution to meet your database's requirements. For more information about selecting a storage solution, see [Review your storage options](#).

Criteria for choosing a data store

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article describes the comparison criteria you should use when evaluating a data store. The goal is to help you determine which data storage types can meet your solution's requirements.

General considerations

Keep the following considerations in mind when making your selection.

Functional requirements

- **Data format.** What type of data are you intending to store? Common types include transactional data, JSON objects, telemetry, search indexes, or flat files.
- **Data size.** How large are the entities you need to store? Will these entities need to be maintained as a single document, or can they be split across multiple documents, tables, collections, and so forth?
- **Scale and structure.** What is the overall amount of storage capacity you need? Do you anticipate partitioning your data?
- **Data relationships.** Will your data need to support one-to-many or many-to-many relationships? Are relationships themselves an important part of the data? Will you need to join or otherwise combine data from within the same dataset, or from external datasets?
- **Consistency model.** How important is it for updates made in one node to appear in other nodes, before further changes can be made? Can you accept eventual consistency? Do you need ACID guarantees for transactions?
- **Schema flexibility.** What kind of schemas will you apply to your data? Will you use a fixed schema, a schema-on-write approach, or a schema-on-read approach?
- **Concurrency.** What kind of concurrency mechanism do you want to use when updating and synchronizing data? Will the application perform many updates that could potentially conflict. If so, you may require record locking and pessimistic concurrency control. Alternatively, can you support optimistic concurrency controls? If so, is simple timestamp-based concurrency control enough, or do you need the added functionality of multi-version concurrency control?
- **Data movement.** Will your solution need to perform ETL tasks to move data to other stores or data warehouses?
- **Data lifecycle.** Is the data write-once, read-many? Can it be moved into cool or cold storage?
- **Other supported features.** Do you need any other specific features, such as schema validation, aggregation, indexing, full-text search, MapReduce, or other query capabilities?

Non-functional requirements

- **Performance and scalability.** What are your data performance requirements? Do you have specific requirements for data ingestion rates and data processing rates? What are the acceptable response times for querying and aggregation of data once ingested? How large will you need the data store to scale up? Is your workload more read-heavy or write-heavy?
- **Reliability.** What overall SLA do you need to support? What level of fault-tolerance do you need to provide for data consumers? What kind of backup and restore capabilities do you need?

- **Replication.** Will your data need to be distributed among multiple replicas or regions? What kind of data replication capabilities do you require?
- **Limits.** Will the limits of a particular data store support your requirements for scale, number of connections, and throughput?

Management and cost

- **Managed service.** When possible, use a managed data service, unless you require specific capabilities that can only be found in an IaaS-hosted data store.
- **Region availability.** For managed services, is the service available in all Azure regions? Does your solution need to be hosted in certain Azure regions?
- **Portability.** Will your data need to be migrated to on-premises, external datacenters, or other cloud hosting environments?
- **Licensing.** Do you have a preference of a proprietary versus OSS license type? Are there any other external restrictions on what type of license you can use?
- **Overall cost.** What is the overall cost of using the service within your solution? How many instances will need to run, to support your uptime and throughput requirements? Consider operations costs in this calculation. One reason to prefer managed services is the reduced operational cost.
- **Cost effectiveness.** Can you partition your data, to store it more cost effectively? For example, can you move large objects out of an expensive relational database into an object store?

Security

- **Security.** What type of encryption do you require? Do you need encryption at rest? What authentication mechanism do you want to use to connect to your data?
- **Auditing.** What kind of audit log do you need to generate?
- **Networking requirements.** Do you need to restrict or otherwise manage access to your data from other network resources? Does data need to be accessible only from inside the Azure environment? Does the data need to be accessible from specific IP addresses or subnets? Does it need to be accessible from applications or services hosted on-premises or in other external datacenters?

DevOps

- **Skill set.** Are there particular programming languages, operating systems, or other technology that your team is particularly adept at using? Are there others that would be difficult for your team to work with?
- **Clients** Is there good client support for your development languages?

Choose a big data storage technology in Azure

3/10/2022 • 8 minutes to read • [Edit Online](#)

This topic compares options for data storage for big data solutions — specifically, data storage for bulk data ingestion and batch processing, as opposed to [analytical data stores](#) or [real-time streaming ingestion](#).

What are your options when choosing data storage in Azure?

There are several options for ingesting data into Azure, depending on your needs.

File storage:

- [Azure Storage blobs](#)
- [Azure Data Lake Store](#)

NoSQL databases:

- [Azure Cosmos DB](#)
- [HBase on HDInsight](#)

Analytical databases:

[Azure Data Explorer](#)

Azure Storage blobs

Azure Storage is a managed storage service that is highly available, secure, durable, scalable, and redundant. Microsoft takes care of maintenance and handles critical problems for you. Azure Storage is the most ubiquitous storage solution Azure provides, due to the number of services and tools that can be used with it.

There are various Azure Storage services you can use to store data. The most flexible option for storing blobs from a number of data sources is [Blob storage](#). Blobs are basically files. They store pictures, documents, HTML files, virtual hard disks (VHDs), big data such as logs, database backups — pretty much anything. Blobs are stored in containers, which are similar to folders. A container provides a grouping of a set of blobs. A storage account can contain an unlimited number of containers, and a container can store an unlimited number of blobs.

Azure Storage is a good choice for big data and analytics solutions, because of its flexibility, high availability, and low cost. It provides hot, cool, and archive storage tiers for different use cases. For more information, see [Azure Blob Storage: Hot, cool, and archive storage tiers](#).

Azure Blob storage can be accessed from Hadoop (available through HDInsight). HDInsight can use a blob container in Azure Storage as the default file system for the cluster. Through a Hadoop distributed file system (HDFS) interface provided by a WASB driver, the full set of components in HDInsight can operate directly on structured or unstructured data stored as blobs. Azure Blob storage can also be accessed via Azure Synapse Analytics using its PolyBase feature.

Other features that make Azure Storage a good choice are:

- [Multiple concurrency strategies](#).
- [Disaster recovery and high availability options](#).
- [Encryption at rest](#).
- [Azure role-based access control \(Azure RBAC\)](#) to control access using Azure Active Directory users and groups.

Azure Data Lake Store

[Azure Data Lake Store](#) is an enterprise-wide hyperscale repository for big data analytic workloads. Data Lake enables you to capture data of any size, type, and ingestion speed in one single [secure](#) location for operational and exploratory analytics.

Data Lake Store does not impose any limits on account sizes, file sizes, or the amount of data that can be stored in a data lake. Data is stored durably by making multiple copies and there is no limit on the duration of time that the data can be stored in the Data Lake. In addition to making multiple copies of files to guard against any unexpected failures, Data lake spreads parts of a file over a number of individual storage servers. This improves the read throughput when reading the file in parallel for performing data analytics.

Data Lake Store can be accessed from Hadoop (available through HDInsight) using the WebHDFS-compatible REST APIs. You may consider using this as an alternative to Azure Storage when your individual or combined file sizes exceed that which is supported by Azure Storage. However, there are [performance tuning guidelines](#) you should follow when using Data Lake Store as your primary storage for an HDInsight cluster, with specific guidelines for [Spark](#), [Hive](#), [MapReduce](#), and [Storm](#). Also, be sure to check Data Lake Store's [regional availability](#), because it is not available in as many regions as Azure Storage, and it needs to be located in the same region as your HDInsight cluster.

Coupled with Azure Data Lake Analytics, Data Lake Store is specifically designed to enable analytics on the stored data and is tuned for performance for data analytics scenarios. Data Lake Store can also be accessed via Azure Synapse using its PolyBase feature.

Azure Cosmos DB

[Azure Cosmos DB](#) is Microsoft's globally distributed multi-model database. Cosmos DB guarantees single-digit-millisecond latencies at the 99th percentile anywhere in the world, offers multiple well-defined consistency models to fine-tune performance, and guarantees high availability with multi-homing capabilities.

Azure Cosmos DB is schema-agnostic. It automatically indexes all the data without requiring you to deal with schema and index management. It's also multi-model, natively supporting document, key-value, graph, and column-family data models.

Azure Cosmos DB features:

- [Geo-replication](#)
- [Elastic scaling of throughput and storage](#) worldwide
- [Five well-defined consistency levels](#)

HBase on HDInsight

[Apache HBase](#) is an open-source, NoSQL database that is built on Hadoop and modeled after Google BigTable. HBase provides random access and strong consistency for large amounts of unstructured and semi-structured data in a schemaless database organized by column families.

Data is stored in the rows of a table, and data within a row is grouped by column family. HBase is schemaless in the sense that neither the columns nor the type of data stored in them need to be defined before using them. The open-source code scales linearly to handle petabytes of data on thousands of nodes. It can rely on data redundancy, batch processing, and other features that are provided by distributed applications in the Hadoop ecosystem.

The [HDInsight implementation](#) leverages the scale-out architecture of HBase to provide automatic sharding of tables, strong consistency for reads and writes, and automatic failover. Performance is enhanced by in-memory caching for reads and high-throughput streaming for writes. In most cases, you'll want to [create the HBase cluster inside a virtual network](#) so other HDInsight clusters and applications can directly access the tables.

Azure Data Explorer

[Azure Data Explorer](#) is a fast and highly scalable data exploration service for log and telemetry data. It helps you handle the many data streams emitted by modern software so you can collect, store, and analyze data. Azure Data Explorer is ideal for analyzing large volumes of diverse data from any data source, such as websites, applications, IoT devices, and more. This data is used for diagnostics, monitoring, reporting, machine learning, and additional analytics capabilities. Azure Data Explorer makes it simple to ingest this data and enables you to do complex ad hoc queries on the data in seconds.

Azure Data Explorer can be linearly [scaled out](#) for increasing ingestion and query processing throughput. An Azure Data Explorer cluster can be [deployed to a Virtual Network](#) for enabling private networks.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need managed, high-speed, cloud-based storage for any type of text or binary data? If yes, then select one of the file storage or analytics options.
- Do you need file storage that is optimized for parallel analytics workloads and high throughput/IOPS? If yes, then choose an option that is tuned to analytics workload performance.
- Do you need to store unstructured or semi-structured data in a schemaless database? If so, select one of the non-relational or analytics options. Compare options for indexing and database models. Depending on the type of data you need to store, the primary database models may be the largest factor.
- Can you use the service in your region? Check the regional availability for each Azure service. See [Products available by region](#).

Capability matrix

The following tables summarize the key differences in capabilities.

File storage capabilities

CAPABILITY	AZURE DATA LAKE STORE	AZURE BLOB STORAGE CONTAINERS
Purpose	Optimized storage for big data analytics workloads	General purpose object store for a wide variety of storage scenarios
Use cases	Batch, streaming analytics, and machine learning data such as log files, IoT data, click streams, large datasets	Any type of text or binary data, such as application back end, backup data, media storage for streaming, and general purpose data
Structure	Hierarchical file system	Object store with flat namespace
Authentication	Based on Azure Active Directory Identities	Based on shared secrets Account Access Keys and Shared Access Signature Keys , and Azure role-based access control (Azure RBAC)
Authentication protocol	OAuth 2.0. Calls must contain a valid JWT (JSON web token) issued by Azure Active Directory	Hash-based message authentication code (HMAC). Calls must contain a Base64-encoded SHA-256 hash over a part of the HTTP request.

Capability	Azure Data Lake Store	Azure Blob Storage Containers
Authorization	POSIX access control lists (ACLs). ACLs based on Azure Active Directory identities can be set file and folder level.	For account-level authorization use Account Access Keys . For account, container, or blob authorization use Shared Access Signature Keys .
Auditing	Available.	Available
Encryption at rest	Transparent, server side	Transparent, server side; Client-side encryption
Developer SDKs	.NET, Java, Python, Node.js	.NET, Java, Python, Node.js, C++, Ruby
Analytics workload performance	Optimized performance for parallel analytics workloads, High Throughput and IOPS	Not optimized for analytics workloads
Size limits	No limits on account sizes, file sizes or number of files	Specific limits documented here
Geo-redundancy	Locally-redundant (LRS), globally redundant (GRS), read-access globally redundant (RA-GRS), zone-redundant (ZRS).	Locally redundant (LRS), globally redundant (GRS), read-access globally redundant (RA-GRS), zone-redundant (ZRS). See here for more information

NoSQL database capabilities

Capability	Azure Cosmos DB	HBase on HDInsight
Primary database model	Document store, graph, key-value store, wide column store	Wide column store
Secondary indexes	Yes	No
SQL language support	Yes	Yes (using the Phoenix JDBC driver)
Consistency	Strong, bounded-staleness, session, consistent prefix, eventual	Strong
Native Azure Functions integration	Yes	No
Automatic global distribution	Yes	No HBase cluster replication can be configured across regions with eventual consistency
Pricing model	Elastically scalable request units (RUs) charged per-second as needed, elastically scalable storage	Per-minute pricing for HDInsight cluster (horizontal scaling of nodes), storage

Analytical database capabilities

Capability	Azure Data Explorer

CAPABILITY	AZURE DATA EXPLORER
Primary database model	Relational (column store), telemetry, and time series store
SQL language support	Yes
Pricing model	Elastically scalable cluster instances
Authentication	Based on Azure Active Directory identities
Encryption at rest	Supported, customer managed keys
Analytics workload performance	Optimized performance for parallel analytics workloads
Size limits	Linearly scalable

Online transaction processing (OLTP)

3/10/2022 • 6 minutes to read • [Edit Online](#)

The management of transactional data using computer systems is referred to as online transaction processing (OLTP). OLTP systems record business interactions as they occur in the day-to-day operation of the organization, and support querying of this data to make inferences.

Transactional data

Transactional data is information that tracks the interactions related to an organization's activities. These interactions are typically business transactions, such as payments received from customers, payments made to suppliers, products moving through inventory, orders taken, or services delivered. Transactional events, which represent the transactions themselves, typically contain a time dimension, some numerical values, and references to other data.

Transactions typically need to be *atomic* and *consistent*. Atomicity means that an entire transaction always succeeds or fails as one unit of work, and is never left in a half-completed state. If a transaction cannot be completed, the database system must roll back any steps that were already done as part of that transaction. In a traditional RDBMS, this rollback happens automatically if a transaction cannot be completed. Consistency means that transactions always leave the data in a valid state. (These are very informal descriptions of atomicity and consistency. There are more formal definitions of these properties, such as [ACID](#).)

Transactional databases can support strong consistency for transactions using various locking strategies, such as pessimistic locking, to ensure that all data is strongly consistent within the context of the enterprise, for all users and processes.

The most common deployment architecture that uses transactional data is the data store tier in a 3-tier architecture. A 3-tier architecture typically consists of a presentation tier, business logic tier, and data store tier. A related deployment architecture is the [N-tier](#) architecture, which may have multiple middle-tiers handling business logic.

Typical traits of transactional data

Transactional data tends to have the following traits:

REQUIREMENT	DESCRIPTION
Normalization	Highly normalized
Schema	Schema on write, strongly enforced
Consistency	Strong consistency, ACID guarantees
Integrity	High integrity
Uses transactions	Yes
Locking strategy	Pessimistic or optimistic
Updateable	Yes

Requirement	Description
Appendable	Yes
Workload	Heavy writes, moderate reads
Indexing	Primary and secondary indexes
Datum size	Small to medium sized
Model	Relational
Data shape	Tabular
Query flexibility	Highly flexible
Scale	Small (MBs) to Large (a few TBs)

When to use this solution

Choose OLTP when you need to efficiently process and store business transactions and immediately make them available to client applications in a consistent way. Use this architecture when any tangible delay in processing would have a negative impact on the day-to-day operations of the business.

OLTP systems are designed to efficiently process and store transactions, as well as query transactional data. The goal of efficiently processing and storing individual transactions by an OLTP system is partly accomplished by data normalization — that is, breaking the data up into smaller chunks that are less redundant. This supports efficiency because it enables the OLTP system to process large numbers of transactions independently, and avoids extra processing needed to maintain data integrity in the presence of redundant data.

Challenges

Implementing and using an OLTP system can create a few challenges:

- OLTP systems are not always good for handling aggregates over large amounts of data, although there are exceptions, such as a well-planned SQL Server-based solution. Analytics against the data, that rely on aggregate calculations over millions of individual transactions, are very resource intensive for an OLTP system. They can be slow to execute and can cause a slow-down by blocking other transactions in the database.
- When conducting analytics and reporting on data that is highly normalized, the queries tend to be complex, because most queries need to de-normalize the data by using joins. Also, naming conventions for database objects in OLTP systems tend to be terse and succinct. The increased normalization coupled with terse naming conventions makes OLTP systems difficult for business users to query, without the help of a DBA or data developer.
- Storing the history of transactions indefinitely and storing too much data in any one table can lead to slow query performance, depending on the number of transactions stored. The common solution is to maintain a relevant window of time (such as the current fiscal year) in the OLTP system and offload historical data to other systems, such as a data mart or [data warehouse](#).

OLTP in Azure

Applications such as websites hosted in [App Service Web Apps](#), REST APIs running in App Service, or mobile or desktop applications communicate with the OLTP system, typically via a REST API intermediary.

In practice, most workloads are not purely OLTP. There tends to be an analytical component as well. In addition, there is an increasing demand for real-time reporting, such as running reports against the operational system. This is also referred to as HTAP (Hybrid Transactional and Analytical Processing). For more information, see [Online Analytical Processing \(OLAP\)](#).

In Azure, all of the following data stores will meet the core requirements for OLTP and the management of transaction data:

- [Azure SQL Database](#)
- [SQL Server in an Azure virtual machine](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Does your solution have specific dependencies for Microsoft SQL Server, MySQL or PostgreSQL compatibility? Your application may limit the data stores you can choose based on the drivers it supports for communicating with the data store, or the assumptions it makes about which database is used.
- Are your write throughput requirements particularly high? If yes, choose an option that provides in-memory tables.
- Is your solution multitenant? If so, consider options that support capacity pools, where multiple database instances draw from an elastic pool of resources, instead of fixed resources per database. This can help you better distribute capacity across all database instances, and can make your solution more cost effective.
- Does your data need to be readable with low latency in multiple regions? If yes, choose an option that supports readable secondary replicas.
- Does your database need to be highly available across geo-graphic regions? If yes, choose an option that supports geographic replication. Also consider the options that support automatic failover from the primary replica to a secondary replica.
- Does your database have specific security needs? If yes, examine the options that provide capabilities like row level security, data masking, and transparent data encryption.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

ABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Is Managed Service	Yes	No	Yes	Yes
Runs on Platform	N/A	Windows, Linux, Docker	N/A	N/A

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Programmability ¹	T-SQL, .NET, R	T-SQL, .NET, R, Python	SQL	SQL, PL/pgSQL

[1] Not including client driver support, which allows many programming languages to connect to and use the OLTP data store.

Scalability capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Maximum database instance size	4 TB	256 TB	16 TB	16 TB
Supports capacity pools	Yes	Yes	No	No
Supports clusters scale out	No	Yes	No	No
Dynamic scalability (scale up)	Yes	No	Yes	Yes

Analytic workload capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Temporal tables	Yes	Yes	No	No
In-memory (memory-optimized) tables	Yes	Yes	No	No
Columnstore support	Yes	Yes	No	No
Adaptive query processing	Yes	Yes	No	No

Availability capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Readable secondaries	Yes	Yes	Yes	Yes
Geographic replication	Yes	Yes	Yes	Yes
Automatic failover to secondary	Yes	No	No	No

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Point-in-time restore	Yes	Yes	Yes	Yes

Security capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Row level security	Yes	Yes	Yes	Yes
Data masking	Yes	Yes	No	No
Transparent data encryption	Yes	Yes	Yes	Yes
Restrict access to specific IP addresses	Yes	Yes	Yes	Yes
Restrict access to allow VNet access only	Yes	Yes	Yes	Yes
Azure Active Directory authentication	Yes	No	Yes	Yes
Active Directory authentication	No	Yes	No	No
Multi-factor authentication	Yes	No	Yes	Yes
Supports Always Encrypted	Yes	Yes	No	No
Private IP	No	Yes	No	No

Choose a data pipeline orchestration technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

Most big data solutions consist of repeated data processing operations, encapsulated in workflows. A pipeline orchestrator is a tool that helps to automate these workflows. An orchestrator can schedule jobs, execute workflows, and coordinate dependencies among tasks.

What are your options for data pipeline orchestration?

In Azure, the following services and tools will meet the core requirements for pipeline orchestration, control flow, and data movement:

- [Azure Data Factory](#)
- [Oozie on HDInsight](#)
- [SQL Server Integration Services \(SSIS\)](#)

These services and tools can be used independently from one another, or used together to create a hybrid solution. For example, the Integration Runtime (IR) in Azure Data Factory V2 can natively execute SSIS packages in a managed Azure compute environment. While there is some overlap in functionality between these services, there are a few key differences.

Key Selection Criteria

To narrow the choices, start by answering these questions:

- Do you need big data capabilities for moving and transforming your data? Usually this means multi-gigabytes to terabytes of data. If yes, then narrow your options to those that best suited for big data.
- Do you require a managed service that can operate at scale? If yes, select one of the cloud-based services that aren't limited by your local processing power.
- Are some of your data sources located on-premises? If yes, look for options that can work with both cloud and on-premises data sources or destinations.
- Is your source data stored in Blob storage on an HDFS filesystem? If so, choose an option that supports Hive queries.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE DATA FACTORY	SQL SERVER INTEGRATION SERVICES (SSIS)	OOZIE ON HDINSIGHT
Managed	Yes	No	Yes
Cloud-based	Yes	No (local)	Yes

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Prerequisite	Azure Subscription	SQL Server	Azure Subscription, HDInsight cluster
Management tools	Azure Portal, PowerShell, CLI, .NET SDK	SSMS, PowerShell	Bash shell, Oozie REST API, Oozie web UI
Pricing	Pay per usage	Licensing / pay for features	No additional charge on top of running the HDInsight cluster

Pipeline capabilities

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Copy data	Yes	Yes	Yes
Custom transformations	Yes	Yes	Yes (MapReduce, Pig, and Hive jobs)
Azure Machine Learning scoring	Yes	Yes (with scripting)	No
HDInsight On-Demand	Yes	No	No
Azure Batch	Yes	No	No
Pig, Hive, MapReduce	Yes	No	Yes
Spark	Yes	No	No
Execute SSIS Package	Yes	Yes	No
Control flow	Yes	Yes	Yes
Access on-premises data	Yes	Yes	No

Scalability capabilities

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Scale up	Yes	No	No
Scale out	Yes	No	Yes (by adding worker nodes to cluster)
Optimized for big data	Yes	No	Yes

Choose a search data store in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article compares technology choices for search data stores in Azure. A search data store is used to create and store specialized indexes for performing searches on free-form text. The text that is indexed may reside in a separate data store, such as blob storage. An application submits a query to the search data store, and the result is a list of matching documents. For more information about this scenario, see [Processing free-form text for search](#).

What are your options when choosing a search data store?

In Azure, all of the following data stores will meet the core requirements for search against free-form text data by providing a search index:

- [Azure Cognitive Search](#)
- [Elasticsearch](#)
- [HDInsight with Solr](#)
- [Azure SQL Database with full text search](#)

Key selection criteria

For search scenarios, begin choosing the appropriate search data store for your needs by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Can you specify your index schema at design time? If not, choose an option that supports updateable schemas.
- Do you need an index only for full-text search, or do you also need rapid aggregation of numeric data and other analytics? If you need functionality beyond full-text search, consider options that support additional analytics.
- Do you need a search index for log analytics, with support for log collection, aggregation, and visualizations on indexed data? If so, consider Elasticsearch, which is part of a log analytics stack.
- Do you need to index data in common document formats such as PDF, Word, PowerPoint, and Excel? If yes, choose an option that provides document indexers.
- Does your database have specific security needs? If yes, consider the security features listed below.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

Capability	Cognitive Search	Elasticsearch	HDInsight with Solr	SQL Database
Is managed service	Yes	No	Yes	Yes

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
REST API	Yes	Yes	Yes	No
Programmability	.NET, Java, Python, JavaScript	Java	Java	T-SQL
Document indexers for common file types (PDF, DOCX, TXT, and so on)	Yes	No	Yes	No

Manageability capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Updateable schema	Yes	Yes	Yes	Yes
Supports scale out	Yes	Yes	Yes	No

Analytic workload capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Supports analytics beyond full text search	No	Yes	Yes	Yes
Part of a log analytics stack	No	Yes (ELK)	No	No
Supports semantic search	Yes (find similar documents only)	Yes	Yes	Yes

Security capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Row-level security	Partial (requires application query to filter by group id)	Partial (requires application query to filter by group id)	Yes	Yes
Transparent data encryption	No	No	No	Yes
Restrict access to specific IP addresses	Yes	Yes	Yes	Yes
Restrict access to allow virtual network access only	Yes	Yes	Yes	Yes

CAPABILITY	COGNITIVE SEARCH	ELASTICSEARCH	HDINSIGHT WITH SOLR	SQL DATABASE
Active Directory authentication (integrated authentication)	No	No	No	Yes

See also

[Processing free-form text for search](#)

Transfer data to and from Azure

3/10/2022 • 7 minutes to read • [Edit Online](#)

There are several options for transferring data to and from Azure, depending on your needs.

Physical transfer

Using physical hardware to transfer data to Azure is a good option when:

- Your network is slow or unreliable.
- Getting additional network bandwidth is cost-prohibitive.
- Security or organizational policies do not allow outbound connections when dealing with sensitive data.

If your primary concern is how long it will take to transfer your data, you may want to run a test to verify whether network transfer is actually slower than physical transport.

There are two main options for physically transporting data to Azure:

- **Azure Import/Export.** The [Azure Import/Export service](#) lets you securely transfer large amounts of data to Azure Blob Storage or Azure Files by shipping internal SATA HDDs or SSDs to an Azure datacenter. You can also use this service to transfer data from Azure Storage to hard disk drives and have these shipped to you for loading on-premises.
- **Azure Data Box.** [Azure Data Box](#) is a Microsoft-provided appliance that works much like the Azure Import/Export service. Microsoft ships you a proprietary, secure, and tamper-resistant transfer appliance and handles the end-to-end logistics, which you can track through the portal. One benefit of the Azure Data Box service is ease of use. You don't need to purchase several hard drives, prepare them, and transfer files to each one. Azure Data Box is supported by a number of industry-leading Azure partners to make it easier to seamlessly use offline transport to the cloud from their products.

Command line tools and APIs

Consider these options when you want scripted and programmatic data transfer.

- **Azure CLI.** The [Azure CLI](#) is a cross-platform tool that allows you to manage Azure services and upload data to Azure Storage.
- **AzCopy.** Use AzCopy from a [Windows](#) or [Linux](#) command-line to easily copy data to and from Azure Blob, File, and Table storage with optimal performance. AzCopy supports concurrency and parallelism, and the ability to resume copy operations when interrupted. You can also use AzCopy to copy data from AWS to Azure. For programmatic access, the [Microsoft Azure Storage Data Movement Library](#) is the core framework that powers AzCopy. It is provided as a .NET Core library.
- **PowerShell.** The `Start-AzureStorageBlobCopy` [PowerShell cmdlet](#) is an option for Windows administrators who are used to PowerShell.
- **AdlCopy.** [AdlCopy](#) enables you to copy data from Azure Storage Blobs into Data Lake Store. It can also be used to copy data between two Azure Data Lake Store accounts. However, it cannot be used to copy data from Data Lake Store to Storage Blobs.
- **Distcp.** If you have an HDInsight cluster with access to Data Lake Store, you can use Hadoop ecosystem tools like [Distcp](#) to copy data to and from an HDInsight cluster storage (WASB) into a Data Lake Store account.

- **Sqoop.** [Sqoop](#) is an Apache project and part of the Hadoop ecosystem. It comes preinstalled on all HDInsight clusters. It allows data transfer between an HDInsight cluster and relational databases such as SQL, Oracle, MySQL, and so on. Sqoop is a collection of related tools, including import and export. Sqoop works with HDInsight clusters using either Azure Storage blobs or Data Lake Store attached storage.
- **PolyBase.** [PolyBase](#) is a technology that accesses data outside of the database through the T-SQL language. In SQL Server 2016, it allows you to run queries on external data in Hadoop or to import/export data from Azure Blob Storage. In Azure Synapse Analytics, you can import/export data from Azure Blob Storage and Azure Data Lake Store. Currently, PolyBase is the fastest method of importing data into Azure Synapse.
- **Hadoop command line.** When you have data that resides on an HDInsight cluster head node, you can use the `hadoop -copyFromLocal` command to copy that data to your cluster's attached storage, such as Azure Storage blob or Azure Data Lake Store. In order to use the Hadoop command, you must first connect to the head node. Once connected, you can upload a file to storage.

Graphical interface

Consider the following options if you are only transferring a few files or data objects and don't need to automate the process.

- **Azure Storage Explorer.** [Azure Storage Explorer](#) is a cross-platform tool that lets you manage the contents of your Azure storage accounts. It allows you to upload, download, and manage blobs, files, queues, tables, and Azure Cosmos DB entities. Use it with Blob storage to manage blobs and folders, as well as upload and download blobs between your local file system and Blob storage, or between storage accounts.
- **Azure portal.** Both Blob storage and Data Lake Store provide a web-based interface for exploring files and uploading new files one at a time. This is a good option if you do not want to install any tools or issue commands to quickly explore your files, or to simply upload a handful of new ones.

Data pipeline

Azure Data Factory. [Azure Data Factory](#) is a managed service best suited for regularly transferring files between a number of Azure services, on-premises, or a combination of the two. Using Azure Data Factory, you can create and schedule data-driven workflows (called pipelines) that ingest data from disparate data stores. It can process and transform the data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning. Create data-driven workflows for [orchestrating](#) and automating data movement and data transformation.

Key Selection Criteria

For data transfer scenarios, choose the appropriate system for your needs by answering these questions:

- Do you need to transfer very large amounts of data, where doing so over an Internet connection would take too long, be unreliable, or too expensive? If yes, consider physical transfer.
- Do you prefer to script your data transfer tasks, so they are reusable? If so, select one of the command line options or Azure Data Factory.
- Do you need to transfer a very large amount of data over a network connection? If so, select an option that is optimized for big data.
- Do you need to transfer data to or from a relational database? If yes, choose an option that supports one or more relational databases. Note that some of these options also require a Hadoop cluster.

- Do you need an automated data pipeline or workflow orchestration? If yes, consider Azure Data Factory.

Capability matrix

The following tables summarize the key differences in capabilities.

Physical transfer

CAPABILITY	AZURE IMPORT/EXPORT SERVICE	AZURE DATA BOX
Form factor	Internal SATA HDDs or SSDs	Secure, tamper-proof, single hardware appliance
Microsoft manages shipping logistics	No	Yes
Integrates with partner products	No	Yes
Custom appliance	No	Yes

Command line tools

Hadoop/HDInsight:

CAPABILITY	DISTCP	SQOOP	HADOOP CLI
Optimized for big data	Yes	Yes	Yes
Copy to relational database	No	Yes	No
Copy from relational database	No	Yes	No
Copy to Blob storage	Yes	Yes	Yes
Copy from Blob storage	Yes	Yes	No
Copy to Data Lake Store	Yes	Yes	Yes
Copy from Data Lake Store	Yes	Yes	No

Other:

CAPABILITY	AZURE CLI	AZCOPY	POWERSHELL	ADLCOPY	POLYBASE
Compatible platforms	Linux, OS X, Windows	Linux, Windows	Windows	Linux, OS X, Windows	SQL Server, Azure Synapse
Optimized for big data	No	Yes	No	Yes ¹	Yes ²
Copy to relational database	No	No	No	No	Yes

CAPABILITY	AZURE CLI	AZCOPY	POWERSHELL	ADLCOPY	POLYBASE
Copy from relational database	No	No	No	No	Yes
Copy to Blob storage	Yes	Yes	Yes	No	Yes
Copy from Blob storage	Yes	Yes	Yes	Yes	Yes
Copy to Data Lake Store	No	Yes	Yes	Yes	Yes
Copy from Data Lake Store	No	No	Yes	Yes	Yes

[1] AdlCopy is optimized for transferring big data when used with a Data Lake Analytics account.

[2] PolyBase [performance can be increased](#) by pushing computation to Hadoop and using [PolyBase scale-out groups](#) to enable parallel data transfer between SQL Server instances and Hadoop nodes.

Graphical interface and Azure Data Factory

CAPABILITY	AZURE STORAGE EXPLORER	AZURE PORTAL *	AZURE DATA FACTORY
Optimized for big data	No	No	Yes
Copy to relational database	No	No	Yes
Copy from relational database	No	No	Yes
Copy to Blob storage	Yes	No	Yes
Copy from Blob storage	Yes	No	Yes
Copy to Data Lake Store	No	No	Yes
Copy from Data Lake Store	No	No	Yes
Upload to Blob storage	Yes	Yes	Yes
Upload to Data Lake Store	Yes	Yes	Yes
Orchestrate data transfers	No	No	Yes
Custom data transformations	No	No	Yes
Pricing model	Free	Free	Pay per usage

* Azure portal in this case means using the web-based exploration tools for Blob storage and Data Lake Store.

Choose an analytical data store in Azure

3/10/2022 • 5 minutes to read • [Edit Online](#)

In a [big data](#) architecture, there is often a need for an analytical data store that serves processed data in a structured format that can be queried using analytical tools. Analytical data stores that support querying of both hot-path and cold-path data are collectively referred to as the serving layer, or data serving storage.

The serving layer deals with processed data from both the hot path and cold path. In the [lambda architecture](#), the serving layer is subdivided into a *speed serving* layer, which stores data that has been processed incrementally, and a *batch serving* layer, which contains the batch-processed output. The serving layer requires strong support for random reads with low latency. Data storage for the speed layer should also support random writes, because batch loading data into this store would introduce undesired delays. On the other hand, data storage for the batch layer does not need to support random writes, but batch writes instead.

There is no single best data management choice for all data storage tasks. Different data management solutions are optimized for different tasks. Most real-world cloud apps and big data processes have a variety of data storage requirements and often use a combination of data storage solutions.

What are your options when choosing an analytical data store?

There are several options for data serving storage in Azure, depending on your needs:

- [Azure Synapse Analytics](#)
- [Azure Synapse Spark pools](#)
- [Azure Databricks](#)
- [Azure Data Explorer](#)
- [Azure SQL Database](#)
- [SQL Server in Azure VM](#)
- [HBase/Phoenix on HDInsight](#)
- [Hive LLAP on HDInsight](#)
- [Azure Analysis Services](#)
- [Azure Cosmos DB](#)

These options provide various database models that are optimized for different types of tasks:

- [Key/value](#) databases hold a single serialized object for each key value. They're good for storing large volumes of data where you want to get one item for a given key value and you don't have to query based on other properties of the item.
- [Document](#) databases are key/value databases in which the values are *documents*. A "document" in this context is a collection of named fields and values. The database typically stores the data in a format such as XML, YAML, JSON, or BSON, but may use plain text. Document databases can query on non-key fields and define secondary indexes to make querying more efficient. This makes a document database more suitable for applications that need to retrieve data based on criteria more complex than the value of the document key. For example, you could query on fields such as product ID, customer ID, or customer name.
- [Column-family](#) databases are key/value data stores that structure data storage into collections of related columns called column families. For example, a census database might have one group of columns for a person's name (first, middle, last), one group for the person's address, and one group for the person's profile information (date of birth, gender). The database can store each column family in a separate partition, while keeping all of the data for one person related to the same key. An application can read a single column family

without reading through all of the data for an entity.

- **Graph** databases store information as a collection of objects and relationships. A graph database can efficiently perform queries that traverse the network of objects and the relationships between them. For example, the objects might be employees in a human resources database, and you might want to facilitate queries such as "find all employees who directly or indirectly work for Scott."
- Telemetry and time series databases are an append-only collection of objects. Telemetry databases efficiently index data in a variety of column stores and in-memory structures, making them the optimal choice for storing and analyzing vast quantities of telemetry and time series data.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need serving storage that can serve as a hot path for your data? If yes, narrow your options to those that are optimized for a speed serving layer.
- Do you need massively parallel processing (MPP) support, where queries are automatically distributed across several processes or nodes? If yes, select an option that supports query scale out.
- Do you prefer to use a relational data store? If so, narrow your options to those with a relational database model. However, note that some non-relational stores support SQL syntax for querying, and tools such as PolyBase can be used to query non-relational data stores.
- Do you collect time series data? Do you use append-only data?

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CapabilitY	SQL DatabasE	Azure Synapse SQL Pool	Azure Synapse Spark Pool	Azure Data Explorer	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Is managed service	Yes	Yes	Yes	Yes	Yes ¹	Yes ¹	Yes	Yes
Primary database model	Relational (columnar format when using columnstore indexes)	Relational tables with columnar storage	Wide column store	Relational (column store), telemetry, and time series store	Wide column store	Hive/In-Memory	Tabular semantic models	Document store, graph, key-value store, wide column store
SQL language support	Yes	Yes	Yes	Yes	Yes (using Phoenix JDBC driver)	Yes	No	Yes

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Optimized for speed serving layer	Yes ²	Yes ³	Yes	Yes	Yes	Yes	No	Yes

[1] With manual configuration and scaling.

[2] Using memory-optimized tables and hash or nonclustered indexes.

[3] Supported as an Azure Stream Analytics output.

Scalability capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Redundant regional servers for high availability	Yes	No	No	Yes	Yes	No	No	Yes
Supports query scale out	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic scalability (scale up)	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Supports in-memory caching of data	Yes	Yes	Yes	Yes	No	Yes	Yes	No

Security capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Authentication	SQL / Azure Active Directory (Azure AD)	SQL / Azure AD	Azure AD	local / Azure AD ¹	local / Azure AD ¹	Azure AD	database users / Azure AD via access control (IAM)

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Data encryption at rest	Yes ²	Yes ²	Yes	Yes ¹	Yes ¹	Yes	Yes
Row-level security	Yes	Yes ³	No	Yes ¹	Yes ¹	Yes	No
Supports firewalls	Yes	Yes	Yes	Yes ⁴	Yes ⁴	Yes	Yes
Dynamic data masking	Yes	Yes	Yes	Yes ¹	Yes	No	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Requires using transparent data encryption (TDE) to encrypt and decrypt your data at rest.

[3] Filter predicates only. See [Row-Level Security](#)

[4] When used within an Azure Virtual Network. See [Extend Azure HDInsight using an Azure Virtual Network](#).

Choose a data analytics technology in Azure

3/10/2022 • 4 minutes to read • [Edit Online](#)

The goal of most big data solutions is to provide insights into the data through analysis and reporting. This can include preconfigured reports and visualizations, or interactive data exploration.

What are your options when choosing a data analytics technology?

There are several options for analysis, visualizations, and reporting in Azure, depending on your needs:

- [Power BI](#)
- [Jupyter Notebooks](#)
- [Zeppelin Notebooks](#)
- [Microsoft Azure Notebooks](#)

Power BI

[Power BI](#) is a suite of business analytics tools. It can connect to hundreds of data sources, and can be used for ad hoc analysis. See [this list](#) of the currently available data sources. Use [Power BI Embedded](#) to integrate Power BI within your own applications without requiring any additional licensing.

Organizations can use Power BI to produce reports and publish them to the organization. Everyone can create personalized dashboards, with governance and [security built in](#). Power BI uses [Azure Active Directory](#) (Azure AD) to authenticate users who log in to the Power BI service, and uses the Power BI login credentials whenever a user attempts to access resources that require authentication.

Jupyter Notebooks

[Jupyter Notebooks](#) provide a browser-based shell that lets data scientists create *notebook* files that contain Python, Scala, or R code and markdown text, making it an effective way to collaborate by sharing and documenting code and results in a single document.

Most varieties of HDInsight clusters, such as Spark or Hadoop, come [preconfigured with Jupyter notebooks](#) for interacting with data and submitting jobs for processing. Depending on the type of HDInsight cluster you are using, one or more kernels will be provided for interpreting and running your code. For example, Spark clusters on HDInsight provide Spark-related kernels that you can select from to execute Python or Scala code using the Spark engine.

Jupyter notebooks provide a great environment for analyzing, visualizing, and processing your data prior to building more advanced visualizations with a BI/reporting tool like Power BI.

Zeppelin Notebooks

[Zeppelin Notebooks](#) are another option for a browser-based shell, similar to Jupyter in functionality. Some HDInsight clusters come [preconfigured with Zeppelin notebooks](#). However, if you are using an [HDInsight Interactive Query](#) (Hive LLAP) cluster, [Zeppelin](#) is currently your only choice of notebook that you can use to run interactive Hive queries. Also, if you are using a [domain-joined HDInsight cluster](#), Zeppelin notebooks are the only type that enables you to assign different user logins to control access to notebooks and the underlying Hive tables.

Microsoft Azure Notebooks

[Azure Notebooks](#) is an online Jupyter Notebooks-based service that enables data scientists to create, run, and share Jupyter Notebooks in cloud-based libraries. Azure Notebooks provides execution environments for Python 2, Python 3, F#, and R, and provides several charting libraries for visualizing your data, such as ggplot,

matplotlib, bokeh, and seaborn.

Unlike Jupyter notebooks running on an HDInsight cluster, which are connected to the cluster's default storage account, Azure Notebooks does not provide any data. You must [load data](#) in a variety of ways, such downloading data from an online source, interacting with Azure Blobs or Table Storage, connecting to a SQL database, or loading data with the Copy Wizard for Azure Data Factory.

Key benefits:

- Free service—no Azure subscription required.
- No need to install Jupyter and the supporting R or Python distributions locally—just use a browser.
- Manage your own online libraries and access them from any device.
- Share your notebooks with collaborators.

Considerations:

- You will be unable to access your notebooks when offline.
- Limited processing capabilities of the free notebook service may not be enough to train large or complex models.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need to connect to numerous data sources, providing a centralized place to create reports for data spread throughout your domain? If so, choose an option that allows you to connect to 100s of data sources.
- Do you want to embed dynamic visualizations in an external website or application? If so, choose an option that provides embedding capabilities.
- Do you want to design your visualizations and reports while offline? If yes, choose an option with offline capabilities.
- Do you need heavy processing power to train large or complex AI models or work with very large data sets? If yes, choose an option that can connect to a big data cluster.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Connect to big data cluster for advanced processing	Yes	Yes	Yes	No
Managed service	Yes	Yes ¹	Yes ¹	Yes
Connect to 100s of data sources	Yes	No	No	No
Offline capabilities	Yes ²	No	No	No

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Embedding capabilities	Yes	No	No	No
Automatic data refresh	Yes	No	No	No
Access to numerous open source packages	No	Yes ³	Yes ³	Yes ⁴
Data transformation/cleaning options	Power Query, R	40 languages, including Python, R, Julia, and Scala	20+ interpreters, including Python, JDBC, and R	Python, F#, R
Pricing	Free for Power BI Desktop (authoring), see pricing for hosting options	Free	Free	Free
Multiuser collaboration	Yes	Yes (through sharing or with a multiuser server like JupyterHub)	Yes	Yes (through sharing)

[1] When used as part of a managed HDInsight cluster.

[2] With the use of Power BI Desktop.

[2] You can search the [Maven repository](#) for community-contributed packages.

[3] Python packages can be installed using either pip or conda. R packages can be installed from CRAN or GitHub. Packages in F# can be installed via nuget.org using the [Paket dependency manager](#).

Choose a batch processing technology in Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

Big data solutions often use long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files from scalable storage (like HDFS, Azure Data Lake Store, and Azure Storage), processing them, and writing the output to new files in scalable storage.

The key requirement of such batch processing engines is the ability to scale out computations, in order to handle a large volume of data. Unlike real-time processing, however, batch processing is expected to have latencies (the time between data ingestion and computing a result) that measure in minutes to hours.

Technology choices for batch processing

Azure Synapse Analytics

[Azure Synapse](#) is a distributed system designed to perform analytics on large data. It supports massive parallel processing (MPP), which makes it suitable for running high-performance analytics. Consider Azure Synapse when you have large amounts of data (more than 1 TB) and are running an analytics workload that will benefit from parallelism.

Azure Data Lake Analytics

[Data Lake Analytics](#) is an on-demand analytics job service. It is optimized for distributed processing of very large data sets stored in Azure Data Lake Store.

- Languages: [U-SQL](#) (including Python, R, and C# extensions).
- Integrates with Azure Data Lake Store, Azure Storage blobs, Azure SQL Database, and Azure Synapse.
- Pricing model is per-job.

HDInsight

HDInsight is a managed Hadoop service. Use it to deploy and manage Hadoop clusters in Azure. For batch processing, you can use [Spark](#), [Hive](#), [Hive LLAP](#), [MapReduce](#).

- Languages: R, Python, Java, Scala, SQL
- Kerberos authentication with Active Directory, Apache Ranger based access control
- Gives you full control of the Hadoop cluster

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform. You can think of it as "Spark as a service." It's the easiest way to use Spark on the Azure platform.

- Languages: R, Python, Java, Scala, Spark SQL
- Fast cluster start times, autotermination, autoscaling.
- Manages the Spark cluster for you.
- Built-in integration with Azure Blob Storage, Azure Data Lake Storage (ADLS), Azure Synapse, and other services. See [Data Sources](#).
- User authentication with Azure Active Directory.
- Web-based [notebooks](#) for collaboration and data exploration.
- Supports [GPU-enabled clusters](#)

Azure Distributed Data Engineering Toolkit

The [Distributed Data Engineering Toolkit](#) (AZTK) is a tool for provisioning on-demand Spark on Docker clusters

in Azure.

AZTK is not an Azure service. Rather, it's a client-side tool with a CLI and Python SDK interface, that's built on Azure Batch. This option gives you the most control over the infrastructure when deploying a Spark cluster.

- Bring your own Docker image.
- Use low-priority VMs for an 80% discount.
- Mixed mode clusters that use both low-priority and dedicated VMs.
- Built in support for Azure Blob Storage and Azure Data Lake connection.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Do you want to author batch processing logic declaratively or imperatively?
- Will you perform batch processing in bursts? If yes, consider options that let you auto-terminate the cluster or whose pricing model is per batch job.
- Do you need to query relational data stores along with your batch processing, for example to look up reference data? If yes, consider the options that enable querying of external relational stores.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT	AZURE DATABRICKS
Is managed service	Yes	Yes	Yes ¹	Yes
Relational data store	Yes	Yes	No	No
Pricing model	Per batch job	By cluster hour	By cluster hour	Databricks Unit ² + cluster hour

[1] With manual configuration.

[2] A Databricks Unit (DBU) is a unit of processing capability per hour.

Capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT WITH SPARK	HDINSIGHT WITH HIVE	HDINSIGHT WITH HIVE LLAP	AZURE DATABRICKS
Autoscaling	No	No	Yes	Yes	Yes	Yes
Scale-out granularity	Per job	Per cluster	Per cluster	Per cluster	Per cluster	Per cluster
In-memory caching of data	No	Yes	Yes	No	Yes	Yes

Capability	Azure Data Lake Analytics	Azure Synapse	HDIInsight with Spark	HDIInsight with Hive	HDIInsight with Hive LLAP	Azure Databricks
Query from external relational stores	Yes	No	Yes	No	No	Yes
Authentication	Azure AD	SQL / Azure AD	No	Azure AD ¹	Azure AD ¹	Azure AD
Auditing	Yes	Yes	No	Yes ¹	Yes ¹	Yes
Row-level security	No	Yes ²	No	Yes ¹	Yes ¹	No
Supports firewalls	Yes	Yes	Yes	Yes ³	Yes ³	No
Dynamic data masking	No	Yes	No	Yes ¹	Yes ¹	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Filter predicates only. See [Row-Level Security](#)

[3] Supported when [used within an Azure Virtual Network](#).

Next steps

- [Analytics architecture design](#)
- [Choose an analytical data store in Azure](#)
- [Choose a data analytics technology in Azure](#)
- [Analytics end-to-end with Azure Synapse](#)

Choose a stream processing technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article compares technology choices for real-time stream processing in Azure.

Real-time stream processing consumes messages from either queue or file-based storage, processes the messages, and forwards the result to another message queue, file store, or database. Processing may include querying, filtering, and aggregating messages. Stream processing engines must be able to consume endless streams of data and produce results with minimal latency. For more information, see [Real time processing](#).

What are your options when choosing a technology for real-time processing?

In Azure, all of the following data stores will meet the core requirements supporting real-time processing:

- [Azure Stream Analytics](#)
- [HDInsight with Spark Streaming](#)
- [Apache Spark in Azure Databricks](#)
- [HDInsight with Storm](#)
- [Azure Functions](#)
- [Azure App Service WebJobs](#)
- [Apache Kafka streams API](#)

Key Selection Criteria

For real-time processing scenarios, begin choosing the appropriate service for your needs by answering these questions:

- Do you prefer a declarative or imperative approach to authoring stream processing logic?
- Do you need built-in support for temporal processing or windowing?
- Does your data arrive in formats besides Avro, JSON, or CSV? If yes, consider options that support any format using custom code.
- Do you need to scale your processing beyond 1 GB/s? If yes, consider the options that scale with the cluster size.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE STREAM ANALYTICS	HDINSIGHT WITH SPARK STREAMING	APACHE SPARK IN AZURE DATABRICKS	HDINSIGHT WITH STORM	AZURE FUNCTIONS	AZURE APP SERVICE WEBJOBS

Capability	Azure Stream Analytics	HdInsight with Spark Streaming	Apache Spark in Azure Databricks	HdInsight with Storm	Azure Functions	Azure App Service WebJobs
Programmability	Stream analytics query language, JavaScript	C#/F#, Java, Python, Scala	C#/F#, Java, Python, R, Scala	C#, Java	C#, F#, Java, Node.js, Python	C#, Java, Node.js, PHP, Python
Programming paradigm	Declarative	Mixture of declarative and imperative	Mixture of declarative and imperative	Imperative	Imperative	Imperative
Pricing model	Streaming units	Per cluster hour	Databricks units	Per cluster hour	Per function execution and resource consumption	Per app service plan hour

Integration capabilities

Capability	Azure Stream Analytics	HdInsight with Spark Streaming	Apache Spark in Azure Databricks	HdInsight with Storm	Azure Functions	Azure App Service WebJobs
Inputs	Azure Event Hubs, Azure IoT Hub, Azure Blob storage	Event Hubs, IoT Hub, Kafka, HDFS, Storage Blobs, Azure Data Lake Store	Event Hubs, IoT Hub, Kafka, HDFS, Storage Blobs, Azure Data Lake Store	Event Hubs, IoT Hub, Storage Blobs, Azure Data Lake Store	Supported bindings	Service Bus, Storage Queues, Storage Blobs, Event Hubs, WebHooks, Cosmos DB, Files
Sinks	Azure Data Lake Store, Azure SQL Database, Storage Blobs, Event Hubs, Power BI, Table Storage, Service Bus Queues, Service Bus Topics, Cosmos DB, Azure Functions	HDFS, Kafka, Storage Blobs, Azure Data Lake Store, Cosmos DB	HDFS, Kafka, Storage Blobs, Azure Data Lake Store, Cosmos DB	Event Hubs, Service Bus, Kafka	Supported bindings	Service Bus, Storage Queues, Storage Blobs, Event Hubs, WebHooks, Cosmos DB, Files

Processing capabilities

Capability	Azure Stream Analytics	HDIInsight with Spark Streaming	Apache Spark in Azure Databricks	HDIInsight with Storm	Azure Functions	Azure App Service WebJobs
Built-in temporal/windowing support	Yes	Yes	Yes	Yes	No	No
Input data formats	Avro, JSON or CSV, UTF-8 encoded	Any format using custom code	Any format using custom code	Any format using custom code	Any format using custom code	Any format using custom code
Scalability	Query partitions	Bounded by cluster size	Bounded by Databricks cluster scale configuration	Bounded by cluster size	Up to 200 function app instances processing in parallel	Bounded by app service plan capacity
Late arrival and out of order event handling support	Yes	Yes	Yes	Yes	No	No

See also:

- [Choosing a real-time message ingestion technology](#)
- [Real time processing](#)

Choose a Microsoft cognitive services technology

3/10/2022 • 3 minutes to read • [Edit Online](#)

Microsoft cognitive services are cloud-based APIs that you can use in artificial intelligence (AI) applications and data flows. They provide you with pretrained models that are ready to use in your application, requiring no data and no model training on your part. The cognitive services are developed by Microsoft's AI and Research team and leverage the latest deep learning algorithms. They are consumed over HTTP REST interfaces. In addition, SDKs are available for many common application development frameworks.

The cognitive services include:

- Text analysis
- Computer vision
- Video analytics
- Speech recognition and generation
- Natural language understanding
- Intelligent search

Key benefits:

- Minimal development effort for state-of-the-art AI services.
- Easy integration into apps via HTTP REST interfaces.
- Built-in support for consuming cognitive services in Azure Data Lake Analytics.

Considerations:

- Only available over the web. Internet connectivity is generally required. An exception is the Custom Vision Service, whose trained model you can export for prediction on devices and at the IoT edge.
- Although considerable customization is supported, the available services may not suit all predictive analytics requirements.

What are your options when choosing amongst the cognitive services?

In Azure, there are dozens of Cognitive Services available. The current listing of these is available in a directory categorized by the functional area they support:

- [Vision](#)
- [Speech](#)
- [Decision](#)
- [Search](#)
- [Language](#)

Key selection criteria

To narrow the choices, start by answering these questions:

- What type of data are you dealing with? Narrow your options based on the type of input data you are working with. For example, if your input is text, select from the services that have an input type of text.

- Do you have the data to train a model? If yes, consider the custom services that enable you to train their underlying models with data that you provide, for improved accuracy and performance.

Capability matrix

The following tables summarize the key differences in capabilities.

Uses prebuilt models

CAPABILITY	INPUT TYPE	KEY BENEFIT
Text Analytics API	Text	Evaluate sentiment and topics to understand what users want.
Entity Linking API	Text	Power your app's data links with named entity recognition and disambiguation.
Language Understanding Intelligent Service (LUIS)	Text	Teach your apps to understand commands from your users.
QnA Maker Service	Text	Distill FAQ formatted information into conversational, easy-to-navigate answers.
Linguistic Analysis API	Text	Simplify complex language concepts and parse text.
Knowledge Exploration Service	Text	Enable interactive search experiences over structured data via natural language inputs.
Web Language Model API	Text	Use predictive language models trained on web-scale data.
Academic Knowledge API	Text	Tap into the wealth of academic content in the Microsoft Academic Graph populated by Bing.
Bing Autosuggest API	Text	Give your app intelligent autosuggest options for searches.
Bing Spell Check API	Text	Detect and correct spelling mistakes in your app.
Translator Text API	Text	Machine translation.
Recommendations API	Text	Predict and recommend items your customers want.
Bing Entity Search API	Text (web search query)	Identify and augment entity information from the web.
Bing Image Search API	Text (web search query)	Search for images.
Bing News Search API	Text (web search query)	Search for news.

CAPABILITY	INPUT TYPE	KEY BENEFIT
Bing Video Search API	Text (web search query)	Search for videos.
Bing Web Search API	Text (web search query)	Get enhanced search details from billions of web documents.
Bing Speech API	Text or Speech	Convert speech to text and back again.
Speaker Recognition API	Speech	Use speech to identify and authenticate individual speakers.
Translator Speech API	Speech	Perform real-time speech translation.
Computer Vision API	Images (or frames from video)	Distill actionable information from images, automatically create description of photos, derive tags, recognize celebrities, extract text, and create accurate thumbnails.
Content Moderator	Text, Images or Video	Automated image, text, and video moderation.
Emotion API	Images (photos with human subjects)	Identify the range emotions of human subjects.
Face API	Images (photos with human subjects)	Detect, identify, analyze, organize, and tag faces in photos.
Video Indexer	Video	Video insights such as sentiment, transcript speech, translate speech, recognize faces and emotions, and extract keywords.

Trained with custom data you provide

CAPABILITY	INPUT TYPE	KEY BENEFIT
Custom Vision Service	Images (or frames from video)	Customize your own computer vision models.
Custom Speech Service	Speech	Overcome speech recognition barriers like speaking style, background noise, and vocabulary.
Custom Decision Service	Web content (for example, RSS feed)	Use machine learning to automatically select the appropriate content for your home page
Bing Custom Search API	Text (web search query)	Commercial-grade search tool.

Compare the machine learning products and technologies from Microsoft

3/10/2022 • 8 minutes to read • [Edit Online](#)

Learn about the machine learning products and technologies from Microsoft. Compare options to help you choose how to most effectively build, deploy, and manage your machine learning solutions.

Cloud-based machine learning products

The following options are available for machine learning in the Azure cloud.

CLOUD OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Azure Machine Learning	Managed platform for machine learning	Use a pretrained model. Or, train, deploy, and manage models on Azure using Python and CLI
Azure Cognitive Services	Pre-built AI capabilities implemented through REST APIs and SDKs	Build intelligent applications quickly using standard programming languages. Doesn't require machine learning and data science expertise
Azure SQL Managed Instance Machine Learning Services	In-database machine learning for SQL	Train and deploy models inside Azure SQL Managed Instance
Machine learning in Azure Synapse Analytics	Analytics service with machine learning	Train and deploy models inside Azure Synapse Analytics
Machine learning and AI with ONNX in Azure SQL Edge	Machine learning in SQL on IoT	Train and deploy models inside Azure SQL Edge
Azure Databricks	Apache Spark-based analytics platform	Build and deploy models and data workflows using integrations with open-source machine learning libraries and the MLFlow platform.

On-premises machine learning products

The following options are available for machine learning on-premises. On-premises servers can also run in a virtual machine in the cloud.

ON-PREMISES OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
SQL Server Machine Learning Services	In-database machine learning for SQL	Train and deploy models inside SQL Server
Machine Learning Services on SQL Server Big Data Clusters	Machine learning in Big Data Clusters	Train and deploy models on SQL Server Big Data Clusters

Development platforms and tools

The following development platforms and tools are available for machine learning.

PLATFORMS/TOOLS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Azure Data Science Virtual Machine	Virtual machine with pre-installed data science tools	Develop machine learning solutions in a pre-configured environment
ML.NET	Open-source, cross-platform machine learning SDK	Develop machine learning solutions for .NET applications
Windows ML	Windows 10 machine learning platform	Evaluate trained models on a Windows 10 device
MMLSpark	Open-source, distributed, machine learning and microservices framework for Apache Spark	Create and deploy scalable machine learning applications for Scala and Python.
Machine Learning extension for Azure Data Studio	Open-source and cross-platform machine learning extension for Azure Data Studio	Manage packages, import machine learning models, make predictions, and create notebooks to run experiments for your SQL databases

Azure Machine Learning

[Azure Machine Learning](#) is a fully managed cloud service used to train, deploy, and manage machine learning models at scale. It fully supports open-source technologies, so you can use tens of thousands of open-source Python packages such as TensorFlow, PyTorch, and scikit-learn. Rich tools are also available, such as [Compute instances](#), [Jupyter notebooks](#), or the [Azure Machine Learning for Visual Studio Code extension](#), a free extension that allows you to manage your resources, model training workflows and deployments in Visual Studio Code. Azure Machine Learning includes features that automate model generation and tuning with ease, efficiency, and accuracy.

Use Python SDK, Jupyter notebooks, R, and the CLI for machine learning at cloud scale. For a low-code or no-code option, use Azure Machine Learning's interactive [designer](#) in the studio to easily and quickly build, test, and deploy models using pre-built machine learning algorithms.

[Try Azure Machine Learning for free.](#)

Type	Cloud-based machine learning solution
Supported languages	Python, R
Machine learning phases	Model training Deployment MLOps/Management
Key benefits	Code first (SDK) and studio & drag-and-drop designer web interface authoring options. Central management of scripts and run history, making it easy to compare model versions. Easy deployment and management of models to the cloud or edge devices.

Considerations	Requires some familiarity with the model management model.
----------------	--

Azure Cognitive Services

[Azure Cognitive Services](#) is a set of *pre-built* APIs that enable you to build apps that use natural methods of communication. The term pre-built suggests that you do not need to bring datasets or data science expertise to train models to use in your applications. That's all done for you and packaged as APIs and SDKs that allow your apps to see, hear, speak, understand, and interpret user needs with just a few lines of code. You can easily add intelligent features to your apps, such as:

- **Vision:** Object detection, face recognition, OCR, etc. See [Computer Vision](#), [Face](#), [Form Recognizer](#).
- **Speech:** Speech-to-text, text-to-speech, speaker recognition, etc. See [Speech Service](#).
- **Language:** Translation, Sentiment analysis, key phrase extraction, language understanding, etc. See [Translator](#), [Text Analytics](#), [Language Understanding](#), [QnA Maker](#)
- **Decision:** Anomaly detection, content moderation, reinforcement learning. See [Anomaly Detector](#), [Content Moderator](#), [Personalizer](#).

Use Cognitive Services to develop apps across devices and platforms. The APIs keep improving, and are easy to set up.

Type	APIs for building intelligent applications
Supported languages	Various options depending on the service. Standard ones are C#, Java, JavaScript, and Python.
Machine learning phases	Deployment
Key benefits	Build intelligent applications using pre-trained models available through REST API and SDK. Variety of models for natural communication methods with vision, speech, language, and decision. No machine learning or data science expertise required.

SQL machine learning

[SQL machine learning](#) adds statistical analysis, data visualization, and predictive analytics in Python and R for relational data, both on-premises and in the cloud. Current platforms and tools include:

- [SQL Server Machine Learning Services](#)
- [Machine Learning Services on SQL Server Big Data Clusters](#)
- [Azure SQL Managed Instance Machine Learning Services](#)
- [Machine learning in Azure Synapse Analytics](#)
- [Machine learning and AI with ONNX in Azure SQL Edge](#)
- [Machine Learning extension for Azure Data Studio](#)

Use SQL machine learning when you need built-in AI and predictive analytics on relational data in SQL.

Type	On-premises predictive analytics for relational data
------	--

Supported languages	Python, R, SQL
Machine learning phases	Data preparation Model training Deployment
Key benefits	Encapsulate predictive logic in a database function, making it easy to include in data-tier logic.
Considerations	Assumes a SQL database as the data tier for your application.

Azure Data Science Virtual Machine

The [Azure Data Science Virtual Machine](#) is a customized virtual machine environment on the Microsoft Azure cloud. It is available in versions for both Windows and Linux Ubuntu. The environment is built specifically for doing data science and developing ML solutions. It has many popular data science, ML frameworks, and other tools pre-installed and pre-configured to jump-start building intelligent applications for advanced analytics.

Use the Data Science VM when you need to run or host your jobs on a single node. Or if you need to remotely scale up your processing on a single machine.

Type	Customized virtual machine environment for data science
Key benefits	<p>Reduced time to install, manage, and troubleshoot data science tools and frameworks.</p> <p>The latest versions of all commonly used tools and frameworks are included.</p> <p>Virtual machine options include highly scalable images with GPU capabilities for intensive data modeling.</p>
Considerations	<p>The virtual machine cannot be accessed when offline.</p> <p>Running a virtual machine incurs Azure charges, so you must be careful to have it running only when required.</p>

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform. Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts. Use Python, R, Scala, and SQL code in web-based notebooks to query, visualize, and model data.

Use Databricks when you want to collaborate on building machine learning solutions on Apache Spark.

Type	Apache Spark-based analytics platform
Supported languages	Python, R, Scala, SQL

Machine learning phases	Data preparation Data preprocessing Model training Model tuning Model inference Management Deployment
--------------------------------	---

ML.NET

[ML.NET](#) is an open-source, and cross-platform machine learning framework. With ML.NET, you can build custom machine learning solutions and integrate them into your .NET applications. ML.NET offers varying levels of interoperability with popular frameworks like TensorFlow and ONNX for training and scoring machine learning and deep learning models. For resource-intensive tasks like training image classification models, you can take advantage of Azure to train your models in the cloud.

Use ML.NET when you want to integrate machine learning solutions into your .NET applications. Choose between the [API](#) for a code-first experience and [Model Builder](#) or the [CLI](#) for a low-code experience.

Type	Open-source cross-platform framework for developing custom machine learning applications with .NET
Languages supported	C#, F#
Machine learning phases	Data preparation Training Deployment
Key benefits	Data science & ML experience not required Use familiar tools (Visual Studio, VS Code) and languages Deploy where .NET runs Extensible Scalable Local-first experience

Windows ML

[Windows ML](#) inference engine allows you to use trained machine learning models in your applications, evaluating trained models locally on Windows 10 devices.

Use Windows ML when you want to use trained machine learning models within your Windows applications.

Type	Inference engine for trained models in Windows devices
Languages supported	C#/C++, JavaScript

MMLSpark

[Microsoft ML for Apache Spark](#) (MMLSpark) is an open-source library that expands the distributed computing framework [Apache Spark](#). MMLSpark adds many deep learning and data science tools to the Spark ecosystem, including seamless integration of [Spark Machine Learning](#) pipelines with [Microsoft Cognitive Toolkit \(CNTK\)](#),

[LightGBM](#), [LIME \(Model Interpretability\)](#), and [OpenCV](#). You can use these tools to create powerful predictive models on any Spark cluster, such as [Azure Databricks](#) or [Cosmic Spark](#).

MMLSpark also brings new networking capabilities to the Spark ecosystem. With the HTTP on Spark project, users can embed any web service into their SparkML models. Additionally, MMLSpark provides easy-to-use tools for orchestrating [Azure Cognitive Services](#) at scale. For production-grade deployment, the Spark Serving project enables high throughput, submillisecond latency web services, backed by your Spark cluster.

Type	Open-source, distributed machine learning and microservices framework for Apache Spark
Languages supported	Scala 2.11, Java, Python 3.5+, R (beta)
Machine learning phases	Data preparation Model training Deployment
Key benefits	Scalability Streaming + Serving compatible Fault-tolerance
Considerations	Requires Apache Spark

Next steps

- To learn about all the Artificial Intelligence (AI) development products available from Microsoft, see [Microsoft AI platform](#).
- For training in developing AI and Machine Learning solutions with Microsoft, see [Microsoft Learn](#).

Choose a real-time message ingestion technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

Real time processing deals with streams of data that are captured in real-time and processed with minimal latency. Many real-time processing solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics.

What are your options for real-time message ingestion?

- [Azure Event Hubs](#)
- [Azure IoT Hub](#)
- [Kafka on HDInsight](#)

Azure Event Hubs

[Azure Event Hubs](#) is a highly scalable data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. Event Hubs provides publish-subscribe capabilities with low latency at massive scale, which makes it appropriate for big data scenarios.

Azure IoT Hub

[Azure IoT Hub](#) is a managed service that enables reliable and secure bidirectional communications between millions of IoT devices and a cloud-based back end.

Features of IoT Hub include:

- Multiple options for device-to-cloud and cloud-to-device communication. These options include one-way messaging, file transfer, and request-reply methods.
- Message routing to other Azure services.
- Queryable store for device metadata and synchronized state information.
- Secure communications and access control using per-device security keys or X.509 certificates.
- Monitoring of device connectivity and device identity management events.

In terms of message ingestion, IoT Hub is similar to Event Hubs. However, it was specifically designed for managing IoT device connectivity, not just message ingestion. For more information, see [Comparison of Azure IoT Hub and Azure Event Hubs](#).

Kafka on HDInsight

[Apache Kafka](#) is an open-source distributed streaming platform that can be used to build real-time data pipelines and streaming applications. Kafka also provides message broker functionality similar to a message queue, where you can publish and subscribe to named data streams. It is horizontally scalable, fault-tolerant, and extremely fast. [Kafka on HDInsight](#) provides a Kafka as a managed, highly scalable, and highly available service in Azure.

Some common use cases for Kafka are:

- **Messaging.** Because it supports the publish-subscribe message pattern, Kafka is often used as a message broker.
- **Activity tracking.** Because Kafka provides in-order logging of records, it can be used to track and re-create activities, such as user actions on a web site.
- **Aggregation.** Using stream processing, you can aggregate information from different streams to combine and centralize the information into operational data.
- **Transformation.** Using stream processing, you can combine and enrich data from multiple input topics into one or more output topics.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need two-way communication between your IoT devices and Azure? If so, choose IoT Hub.
- Do you need to manage access for individual devices and be able to revoke access to a specific device? If yes, choose IoT Hub.

Capability matrix

The following tables summarize the key differences in capabilities.

CAPABILITY	IOT HUB	EVENT HUBS	KAFKA ON HDINSIGHT
Cloud-to-device communications	Yes	No	No
Device-initiated file upload	Yes	No	No
Device state information	Device twins	No	No
Protocol support	MQTT, AMQP, HTTPS ¹	AMQP, HTTPS, Kafka Protocol	Kafka Protocol
Security	Per-device identity; revocable access control.	Shared access policies; limited revocation through publisher policies.	Authentication using SASL; pluggable authorization; integration with external authentication services supported.

[1] You can also use [Azure IoT protocol gateway](#) as a custom gateway to enable protocol adaptation for IoT Hub.

For more information, see [Comparison of Azure IoT Hub and Azure Event Hubs](#).

Best practices in cloud applications

3/10/2022 • 3 minutes to read • [Edit Online](#)

These best practices can help you build reliable, scalable, and secure applications in the cloud. They offer guidelines and tips for designing and implementing efficient and robust systems, mechanisms, and approaches. Many also include code examples that you can use with Azure services. The practices apply to any distributed system, whether your host is Azure or a different cloud platform.

Catalog of practices

This table lists various best practices. The **Related pillars or patterns** column contains the following links:

- Cloud development challenges that the practice and related design patterns address.
- Pillars of the [Microsoft Azure Well-Architected Framework](#) that the practice focuses on.

PRACTICE	SUMMARY	RELATED PILLARS OR PATTERNS
API design	<p>Design web APIs to support platform independence by using standard protocols and agreed-upon data formats. Promote service evolution so that clients can discover functionality without requiring modification.</p> <p>Improve response times and prevent transient faults by supporting partial responses and providing ways to filter and paginate data.</p>	Design and implementation , Performance efficiency , Operational excellence
API implementation	<p>Implement web APIs to be efficient, responsive, scalable, and available.</p> <p>Make actions idempotent, support content negotiation, and follow the HTTP specification. Handle exceptions, and support the discovery of resources.</p> <p>Provide ways to handle large requests and minimize network traffic.</p>	Design and implementation , Operational excellence
Autoscaling	<p>Design apps to dynamically allocate and de-allocate resources to satisfy performance requirements and minimize costs.</p> <p>Take advantage of Azure Monitor autoscale and the built-in autoscaling that many Azure components offer.</p>	Performance efficiency , Cost optimization
Background jobs	<p>Implement batch jobs, processing tasks, and workflows as background jobs.</p> <p>Use Azure platform services to host these tasks. Trigger tasks with events or schedules, and return results to calling tasks.</p>	Design and implementation , Operational excellence

PRACTICE	SUMMARY	RELATED PILLARS OR PATTERNS
Caching	Improve performance by copying data to fast storage that's close to apps. Cache data that you read often but rarely modify. Manage data expiration and concurrency. See how to populate caches and use the Azure Cache for Redis service.	Data management, Performance efficiency
Content delivery network	Use content delivery networks (CDNs) to efficiently deliver web content to users and reduce load on web apps. Overcome deployment, versioning, security, and resilience challenges.	Data management, Performance efficiency
Data partitioning	Partition data to improve scalability, availability, and performance, and to reduce contention and data storage costs. Use horizontal, vertical, and functional partitioning in efficient ways.	Data management, Performance efficiency, Cost optimization
Data partitioning strategies (by service)	Partition data in Azure SQL Database and Azure Storage services like Azure Table Storage and Azure Blob Storage . Shard your data to distribute loads, reduce latency, and support horizontal scaling.	Data management, Performance efficiency, Cost optimization
Host name preservation	Learn why it's important to preserve the original HTTP host name between a reverse proxy and its back-end web application, and how to implement this recommendation for the most common Azure services.	Design and implementation, Reliability
Message encoding considerations	Use asynchronous messages to exchange information between system components. Choose the payload structure, encoding format, and serialization library that work best with your data.	Messaging, Security
Monitoring and diagnostics	Track system health, usage, and performance with a monitoring and diagnostics pipeline. Turn monitoring data into alerts, reports, and triggers that help in various situations. Examples include detecting and correcting issues, spotting potential problems, meeting performance guarantees, and fulfilling auditing requirements.	Operational excellence
Retry guidance for specific services	Use, adapt, and extend the retry mechanisms that Azure services and client SDKs offer. Develop a systematic and robust approach for managing temporary issues with connections, operations, and resources.	Design and implementation, Reliability

PRACTICE	SUMMARY	RELATED PILLARS OR PATTERNS
Transient fault handling	Handle transient faults caused by unavailable networks or resources. Overcome challenges when developing appropriate retry strategies. Avoid duplicating layers of retry code and other anti-patterns.	Design and implementation, Reliability

Next steps

- [Web API design](#)
- [Web API implementation](#)

Related resources

- [Cloud design patterns](#)
- [Microsoft Azure Well-Architected Framework](#)

RESTful web API design

3/10/2022 • 28 minutes to read • [Edit Online](#)

Most modern web applications expose APIs that clients can use to interact with the application. A well-designed web API should aim to support:

- **Platform independence.** Any client should be able to call the API, regardless of how the API is implemented internally. This requires using standard protocols, and having a mechanism whereby the client and the web service can agree on the format of the data to exchange.
- **Service evolution.** The web API should be able to evolve and add functionality independently from client applications. As the API evolves, existing client applications should continue to function without modification. All functionality should be discoverable so that client applications can fully use it.

This guidance describes issues that you should consider when designing a web API.

What is REST?

In 2000, Roy Fielding proposed Representational State Transfer (REST) as an architectural approach to designing web services. REST is an architectural style for building distributed systems based on hypermedia. REST is independent of any underlying protocol and is not necessarily tied to HTTP. However, most common REST API implementations use HTTP as the application protocol, and this guide focuses on designing REST APIs for HTTP.

A primary advantage of REST over HTTP is that it uses open standards, and does not bind the implementation of the API or the client applications to any specific implementation. For example, a REST web service could be written in ASP.NET, and client applications can use any language or toolset that can generate HTTP requests and parse HTTP responses.

Here are some of the main design principles of RESTful APIs using HTTP:

- REST APIs are designed around *resources*, which are any kind of object, data, or service that can be accessed by the client.
- A resource has an *identifier*, which is a URI that uniquely identifies that resource. For example, the URI for a particular customer order might be:

```
https://adventure-works.com/orders/1
```

- Clients interact with a service by exchanging *representations* of resources. Many web APIs use JSON as the exchange format. For example, a GET request to the URI listed above might return this response body:

```
{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}
```

- REST APIs use a uniform interface, which helps to decouple the client and service implementations. For REST APIs built on HTTP, the uniform interface includes using standard HTTP verbs to perform operations on resources. The most common operations are GET, POST, PUT, PATCH, and DELETE.
- REST APIs use a stateless request model. HTTP requests should be independent and may occur in any order, so keeping transient state information between requests is not feasible. The only place where information is stored is in the resources themselves, and each request should be an atomic operation. This constraint enables web services to be highly scalable, because there is no need to retain any affinity

between clients and specific servers. Any server can handle any request from any client. That said, other factors can limit scalability. For example, many web services write to a backend data store, which may be hard to scale out. For more information about strategies to scale out a data store, see [Horizontal, vertical, and functional data partitioning](#).

- REST APIs are driven by hypermedia links that are contained in the representation. For example, the following shows a JSON representation of an order. It contains links to get or update the customer associated with the order.

```
{  
    "orderID":3,  
    "productID":2,  
    "quantity":4,  
    "orderValue":16.60,  
    "links": [  
        {"rel":"product","href":"https://adventure-works.com/customers/3", "action":"GET"},  
        {"rel":"product","href":"https://adventure-works.com/customers/3", "action":"PUT"}  
    ]  
}
```

In 2008, Leonard Richardson proposed the following [maturity model](#) for web APIs:

- Level 0: Define one URI, and all operations are POST requests to this URI.
- Level 1: Create separate URIs for individual resources.
- Level 2: Use HTTP methods to define operations on resources.
- Level 3: Use hypermedia (HATEOAS, described below).

Level 3 corresponds to a truly RESTful API according to Fielding's definition. In practice, many published web APIs fall somewhere around level 2.

Organize the API design around resources

Focus on the business entities that the web API exposes. For example, in an e-commerce system, the primary entities might be customers and orders. Creating an order can be achieved by sending an HTTP POST request that contains the order information. The HTTP response indicates whether the order was placed successfully or not. When possible, resource URIs should be based on nouns (the resource) and not verbs (the operations on the resource).

```
https://adventure-works.com/orders // Good  
https://adventure-works.com/create-order // Avoid
```

A resource doesn't have to be based on a single physical data item. For example, an order resource might be implemented internally as several tables in a relational database, but presented to the client as a single entity. Avoid creating APIs that simply mirror the internal structure of a database. The purpose of REST is to model entities and the operations that an application can perform on those entities. A client should not be exposed to the internal implementation.

Entities are often grouped together into collections (orders, customers). A collection is a separate resource from the item within the collection, and should have its own URI. For example, the following URI might represent the collection of orders:

```
https://adventure-works.com/orders
```

Sending an HTTP GET request to the collection URI retrieves a list of items in the collection. Each item in the

collection also has its own unique URI. An HTTP GET request to the item's URI returns the details of that item.

Adopt a consistent naming convention in URIs. In general, it helps to use plural nouns for URIs that reference collections. It's a good practice to organize URIs for collections and items into a hierarchy. For example,

/customers is the path to the customers collection, and /customers/5 is the path to the customer with ID equal to 5. This approach helps to keep the web API intuitive. Also, many web API frameworks can route requests based on parameterized URI paths, so you could define a route for the path /customers/{id}.

Also consider the relationships between different types of resources and how you might expose these associations. For example, the /customers/5/orders might represent all of the orders for customer 5. You could also go in the other direction, and represent the association from an order back to a customer with a URI such as /orders/99/customer. However, extending this model too far can become cumbersome to implement. A better solution is to provide navigable links to associated resources in the body of the HTTP response message. This mechanism is described in more detail in the section [Use HATEOAS to enable navigation to related resources](#).

In more complex systems, it can be tempting to provide URIs that enable a client to navigate through several levels of relationships, such as /customers/1/orders/99/products. However, this level of complexity can be difficult to maintain and is inflexible if the relationships between resources change in the future. Instead, try to keep URIs relatively simple. Once an application has a reference to a resource, it should be possible to use this reference to find items related to that resource. The preceding query can be replaced with the URI /customers/1/orders to find all the orders for customer 1, and then /orders/99/products to find the products in this order.

TIP

Avoid requiring resource URIs more complex than *collection/item/collection*.

Another factor is that all web requests impose a load on the web server. The more requests, the bigger the load. Therefore, try to avoid "chatty" web APIs that expose a large number of small resources. Such an API may require a client application to send multiple requests to find all of the data that it requires. Instead, you might want to denormalize the data and combine related information into bigger resources that can be retrieved with a single request. However, you need to balance this approach against the overhead of fetching data that the client doesn't need. Retrieving large objects can increase the latency of a request and incur additional bandwidth costs. For more information about these performance antipatterns, see [Chatty I/O](#) and [Extraneous Fetching](#).

Avoid introducing dependencies between the web API and the underlying data sources. For example, if your data is stored in a relational database, the web API doesn't need to expose each table as a collection of resources. In fact, that's probably a poor design. Instead, think of the web API as an abstraction of the database. If necessary, introduce a mapping layer between the database and the web API. That way, client applications are isolated from changes to the underlying database scheme.

Finally, it might not be possible to map every operation implemented by a web API to a specific resource. You can handle such *non-resource* scenarios through HTTP requests that invoke a function and return the results as an HTTP response message. For example, a web API that implements simple calculator operations such as add and subtract could provide URIs that expose these operations as pseudo resources and use the query string to specify the parameters required. For example, a GET request to the URI /add?operand1=99&operand2=1 would return a response message with the body containing the value 100. However, only use these forms of URIs sparingly.

Define API operations in terms of HTTP methods

The HTTP protocol defines a number of methods that assign semantic meaning to a request. The common HTTP methods used by most RESTful web APIs are:

- **GET** retrieves a representation of the resource at the specified URI. The body of the response message contains the details of the requested resource.
- **POST** creates a new resource at the specified URI. The body of the request message provides the details of the new resource. Note that POST can also be used to trigger operations that don't actually create resources.
- **PUT** either creates or replaces the resource at the specified URI. The body of the request message specifies the resource to be created or updated.
- **PATCH** performs a partial update of a resource. The request body specifies the set of changes to apply to the resource.
- **DELETE** removes the resource at the specified URI.

The effect of a specific request should depend on whether the resource is a collection or an individual item. The following table summarizes the common conventions adopted by most RESTful implementations using the e-commerce example. Not all of these requests might be implemented—it depends on the specific scenario.

RESOURCE	POST	GET	PUT	DELETE
/customers	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
/customers/1	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exists	Remove customer 1
/customers/1/orders	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

The differences between POST, PUT, and PATCH can be confusing.

- A POST request creates a resource. The server assigns a URI for the new resource, and returns that URI to the client. In the REST model, you frequently apply POST requests to collections. The new resource is added to the collection. A POST request can also be used to submit data for processing to an existing resource, without any new resource being created.
- A PUT request creates a resource *or* updates an existing resource. The client specifies the URI for the resource. The request body contains a complete representation of the resource. If a resource with this URI already exists, it is replaced. Otherwise a new resource is created, if the server supports doing so. PUT requests are most frequently applied to resources that are individual items, such as a specific customer, rather than collections. A server might support updates but not creation via PUT. Whether to support creation via PUT depends on whether the client can meaningfully assign a URI to a resource before it exists. If not, then use POST to create resources and PUT or PATCH to update.
- A PATCH request performs a *partial update* to an existing resource. The client specifies the URI for the resource. The request body specifies a set of *changes* to apply to the resource. This can be more efficient than using PUT, because the client only sends the changes, not the entire representation of the resource. Technically PATCH can also create a new resource (by specifying a set of updates to a "null" resource), if the server supports this.

PUT requests must be idempotent. If a client submits the same PUT request multiple times, the results should always be the same (the same resource will be modified with the same values). POST and PATCH requests are not guaranteed to be idempotent.

Conform to HTTP semantics

This section describes some typical considerations for designing an API that conforms to the HTTP specification. However, it doesn't cover every possible detail or scenario. When in doubt, consult the HTTP specifications.

Media types

As mentioned earlier, clients and servers exchange representations of resources. For example, in a POST request, the request body contains a representation of the resource to create. In a GET request, the response body contains a representation of the fetched resource.

In the HTTP protocol, formats are specified through the use of *media types*, also called MIME types. For non-binary data, most web APIs support JSON (media type = application/json) and possibly XML (media type = application/xml).

The Content-Type header in a request or response specifies the format of the representation. Here is an example of a POST request that includes JSON data:

```
POST https://adventure-works.com/orders HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 57

{"Id":1,"Name":"Gizmo","Category":"Widgets","Price":1.99}
```

If the server doesn't support the media type, it should return HTTP status code 415 (Unsupported Media Type).

A client request can include an Accept header that contains a list of media types the client will accept from the server in the response message. For example:

```
GET https://adventure-works.com/orders/2 HTTP/1.1
Accept: application/json
```

If the server cannot match any of the media type(s) listed, it should return HTTP status code 406 (Not Acceptable).

GET methods

A successful GET method typically returns HTTP status code 200 (OK). If the resource cannot be found, the method should return 404 (Not Found).

If the request was fulfilled but there is no response body included in the HTTP response, then it should return HTTP status code 204 (No Content); for example, a search operation yielding no matches might be implemented with this behavior.

POST methods

If a POST method creates a new resource, it returns HTTP status code 201 (Created). The URI of the new resource is included in the Location header of the response. The response body contains a representation of the resource.

If the method does some processing but does not create a new resource, the method can return HTTP status code 200 and include the result of the operation in the response body. Alternatively, if there is no result to return, the method can return HTTP status code 204 (No Content) with no response body.

If the client puts invalid data into the request, the server should return HTTP status code 400 (Bad Request). The response body can contain additional information about the error or a link to a URI that provides more details.

PUT methods

If a PUT method creates a new resource, it returns HTTP status code 201 (Created), as with a POST method. If the method updates an existing resource, it returns either 200 (OK) or 204 (No Content). In some cases, it might not be possible to update an existing resource. In that case, consider returning HTTP status code 409 (Conflict).

Consider implementing bulk HTTP PUT operations that can batch updates to multiple resources in a collection. The PUT request should specify the URI of the collection, and the request body should specify the details of the

resources to be modified. This approach can help to reduce chattiness and improve performance.

PATCH methods

With a PATCH request, the client sends a set of updates to an existing resource, in the form of a *patch document*. The server processes the patch document to perform the update. The patch document doesn't describe the whole resource, only a set of changes to apply. The specification for the PATCH method ([RFC 5789](#)) doesn't define a particular format for patch documents. The format must be inferred from the media type in the request.

JSON is probably the most common data format for web APIs. There are two main JSON-based patch formats, called *JSON patch* and *JSON merge patch*.

JSON merge patch is somewhat simpler. The patch document has the same structure as the original JSON resource, but includes just the subset of fields that should be changed or added. In addition, a field can be deleted by specifying `null` for the field value in the patch document. (That means merge patch is not suitable if the original resource can have explicit null values.)

For example, suppose the original resource has the following JSON representation:

```
{  
  "name": "gizmo",  
  "category": "widgets",  
  "color": "blue",  
  "price": 10  
}
```

Here is a possible JSON merge patch for this resource:

```
{  
  "price": 12,  
  "color": null,  
  "size": "small"  
}
```

This tells the server to update `price`, delete `color`, and add `size`, while `name` and `category` are not modified.

For the exact details of JSON merge patch, see [RFC 7396](#). The media type for JSON merge patch is

`application/merge-patch+json`.

Merge patch is not suitable if the original resource can contain explicit null values, due to the special meaning of `null` in the patch document. Also, the patch document doesn't specify the order that the server should apply the updates. That may or may not matter, depending on the data and the domain. JSON patch, defined in [RFC 6902](#), is more flexible. It specifies the changes as a sequence of operations to apply. Operations include add, remove, replace, copy, and test (to validate values). The media type for JSON patch is

`application/json-patch+json`.

Here are some typical error conditions that might be encountered when processing a PATCH request, along with the appropriate HTTP status code.

ERROR CONDITION	HTTP STATUS CODE
The patch document format isn't supported.	415 (Unsupported Media Type)
Malformed patch document.	400 (Bad Request)
The patch document is valid, but the changes can't be applied to the resource in its current state.	409 (Conflict)

DELETE methods

If the delete operation is successful, the web server should respond with HTTP status code 204 (No Content), indicating that the process has been successfully handled, but that the response body contains no further information. If the resource doesn't exist, the web server can return HTTP 404 (Not Found).

Asynchronous operations

Sometimes a POST, PUT, PATCH, or DELETE operation might require processing that takes a while to complete. If you wait for completion before sending a response to the client, it may cause unacceptable latency. If so, consider making the operation asynchronous. Return HTTP status code 202 (Accepted) to indicate the request was accepted for processing but is not completed.

You should expose an endpoint that returns the status of an asynchronous request, so the client can monitor the status by polling the status endpoint. Include the URI of the status endpoint in the Location header of the 202 response. For example:

```
HTTP/1.1 202 Accepted
Location: /api/status/12345
```

If the client sends a GET request to this endpoint, the response should contain the current status of the request. Optionally, it could also include an estimated time to completion or a link to cancel the operation.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "In progress",
  "link": { "rel": "cancel", "method": "delete", "href": "/api/status/12345" }
}
```

If the asynchronous operation creates a new resource, the status endpoint should return status code 303 (See Other) after the operation completes. In the 303 response, include a Location header that gives the URI of the new resource:

```
HTTP/1.1 303 See Other
Location: /api/orders/12345
```

For more information, see [Asynchronous Request-Reply pattern](#).

Filter and paginate data

Exposing a collection of resources through a single URI can lead to applications fetching large amounts of data when only a subset of the information is required. For example, suppose a client application needs to find all orders with a cost over a specific value. It might retrieve all orders from the `/orders` URI and then filter these orders on the client side. Clearly this process is highly inefficient. It wastes network bandwidth and processing power on the server hosting the web API.

Instead, the API can allow passing a filter in the query string of the URI, such as `/orders?minCost=n`. The web API is then responsible for parsing and handling the `minCost` parameter in the query string and returning the filtered results on the server side.

GET requests over collection resources can potentially return a large number of items. You should design a web API to limit the amount of data returned by any single request. Consider supporting query strings that specify the maximum number of items to retrieve and a starting offset into the collection. For example:

```
/orders?limit=25&offset=50
```

Also consider imposing an upper limit on the number of items returned, to help prevent Denial of Service attacks. To assist client applications, GET requests that return paginated data should also include some form of metadata that indicate the total number of resources available in the collection.

You can use a similar strategy to sort data as it is fetched, by providing a sort parameter that takes a field name as the value, such as `/orders?sort=ProductID`. However, this approach can have a negative effect on caching, because query string parameters form part of the resource identifier used by many cache implementations as the key to cached data.

You can extend this approach to limit the fields returned for each item, if each item contains a large amount of data. For example, you could use a query string parameter that accepts a comma-delimited list of fields, such as `/orders?fields=ProductID,Quantity`.

Give all optional parameters in query strings meaningful defaults. For example, set the `limit` parameter to 10 and the `offset` parameter to 0 if you implement pagination, set the sort parameter to the key of the resource if you implement ordering, and set the `fields` parameter to all fields in the resource if you support projections.

Support partial responses for large binary resources

A resource may contain large binary fields, such as files or images. To overcome problems caused by unreliable and intermittent connections and to improve response times, consider enabling such resources to be retrieved in chunks. To do this, the web API should support the Accept-Ranges header for GET requests for large resources. This header indicates that the GET operation supports partial requests. The client application can submit GET requests that return a subset of a resource, specified as a range of bytes.

Also, consider implementing HTTP HEAD requests for these resources. A HEAD request is similar to a GET request, except that it only returns the HTTP headers that describe the resource, with an empty message body. A client application can issue a HEAD request to determine whether to fetch a resource by using partial GET requests. For example:

```
HEAD https://adventure-works.com/products/10?fields=productImage HTTP/1.1
```

Here is an example response message:

```
HTTP/1.1 200 OK
```

```
Accept-Ranges: bytes
Content-Type: image/jpeg
Content-Length: 4580
```

The Content-Length header gives the total size of the resource, and the Accept-Ranges header indicates that the corresponding GET operation supports partial results. The client application can use this information to retrieve the image in smaller chunks. The first request fetches the first 2500 bytes by using the Range header:

```
GET https://adventure-works.com/products/10?fields=productImage HTTP/1.1
Range: bytes=0-2499
```

The response message indicates that this is a partial response by returning HTTP status code 206. The Content-Length header specifies the actual number of bytes returned in the message body (not the size of the resource), and the Content-Range header indicates which part of the resource this is (bytes 0-2499 out of 4580):

```
HTTP/1.1 206 Partial Content
```

```
Accept-Ranges: bytes
Content-Type: image/jpeg
Content-Length: 2500
Content-Range: bytes 0-2499/4580
```

```
[...]
```

A subsequent request from the client application can retrieve the remainder of the resource.

Use HATEOAS to enable navigation to related resources

One of the primary motivations behind REST is that it should be possible to navigate the entire set of resources without requiring prior knowledge of the URI scheme. Each HTTP GET request should return the information necessary to find the resources related directly to the requested object through hyperlinks included in the response, and it should also be provided with information that describes the operations available on each of these resources. This principle is known as HATEOAS, or Hypertext as the Engine of Application State. The system is effectively a finite state machine, and the response to each request contains the information necessary to move from one state to another; no other information should be necessary.

NOTE

Currently there are no general-purpose standards that define how to model the HATEOAS principle. The examples shown in this section illustrate one possible, proprietary solution.

For example, to handle the relationship between an order and a customer, the representation of an order could include links that identify the available operations for the customer of the order. Here is a possible representation:

```
{
  "orderID":3,
  "productID":2,
  "quantity":4,
  "orderValue":16.60,
  "links": [
    {
      "rel": "customer",
      "href": "https://adventure-works.com/customers/3",
      "action": "GET",
      "types": ["text/xml", "application/json"]
    },
    {
      "rel": "customer",
      "href": "https://adventure-works.com/customers/3",
      "action": "PUT",
      "types": ["application/x-www-form-urlencoded"]
    },
    {
      "rel": "customer",
      "href": "https://adventure-works.com/customers/3",
      "action": "DELETE",
      "types": []
    },
    {
      "rel": "self",
      "href": "https://adventure-works.com/orders/3",
      "action": "GET",
      "types": ["text/xml", "application/json"]
    },
    {
      "rel": "self",
      "href": "https://adventure-works.com/orders/3",
      "action": "PUT",
      "types": ["application/x-www-form-urlencoded"]
    },
    {
      "rel": "self",
      "href": "https://adventure-works.com/orders/3",
      "action": "DELETE",
      "types": []
    }
  ]
}
```

In this example, the `links` array has a set of links. Each link represents an operation on a related entity. The data for each link includes the relationship ("customer"), the URI (`https://adventure-works.com/customers/3`), the HTTP method, and the supported MIME types. This is all the information that a client application needs to be able to invoke the operation.

The `links` array also includes self-referencing information about the resource itself that has been retrieved. These have the relationship *self*.

The set of links that are returned may change, depending on the state of the resource. This is what is meant by hypertext being the "engine of application state."

Versioning a RESTful web API

It is highly unlikely that a web API will remain static. As business requirements change new collections of resources may be added, the relationships between resources might change, and the structure of the data in resources might be amended. While updating a web API to handle new or differing requirements is a relatively straightforward process, you must consider the effects that such changes will have on client applications consuming the web API. The issue is that although the developer designing and implementing a web API has full

control over that API, the developer does not have the same degree of control over client applications, which may be built by third-party organizations operating remotely. The primary imperative is to enable existing client applications to continue functioning unchanged while allowing new client applications to take advantage of new features and resources.

Versioning enables a web API to indicate the features and resources that it exposes, and a client application can submit requests that are directed to a specific version of a feature or resource. The following sections describe several different approaches, each of which has its own benefits and trade-offs.

No versioning

This is the simplest approach, and may be acceptable for some internal APIs. Significant changes could be represented as new resources or new links. Adding content to existing resources might not present a breaking change as client applications that are not expecting to see this content will ignore it.

For example, a request to the URI `https://adventure-works.com/customers/3` should return the details of a single customer containing `id`, `name`, and `address` fields expected by the client application:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","address":"1 Microsoft Way Redmond WA 98053"}
```

NOTE

For simplicity, the example responses shown in this section do not include HATEOAS links.

If the `DateCreated` field is added to the schema of the customer resource, then the response would look like this:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":"1 Microsoft Way Redmond WA 98053"}
```

Existing client applications might continue functioning correctly if they are capable of ignoring unrecognized fields, while new client applications can be designed to handle this new field. However, if more radical changes to the schema of resources occur (such as removing or renaming fields) or the relationships between resources change then these may constitute breaking changes that prevent existing client applications from functioning correctly. In these situations, you should consider one of the following approaches.

URI versioning

Each time you modify the web API or change the schema of resources, you add a version number to the URI for each resource. The previously existing URIs should continue to operate as before, returning resources that conform to their original schema.

Extending the previous example, if the `address` field is restructured into subfields containing each constituent part of the address (such as `streetAddress`, `city`, `state`, and `zipCode`), this version of the resource could be exposed through a URI containing a version number, such as `https://adventure-works.com/v2/customers/3`:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":{"streetAddress":"1 Microsoft Way","city":"Redmond","state":"WA","zipCode":98053}}
```

This versioning mechanism is very simple but depends on the server routing the request to the appropriate endpoint. However, it can become unwieldy as the web API matures through several iterations and the server has to support a number of different versions. Also, from a purist's point of view, in all cases the client applications are fetching the same data (customer 3), so the URI should not really be different depending on the version. This scheme also complicates implementation of HATEOAS as all links will need to include the version number in their URIs.

Query string versioning

Rather than providing multiple URIs, you can specify the version of the resource by using a parameter within the query string appended to the HTTP request, such as `https://adventure-works.com/customers/3?version=2`. The version parameter should default to a meaningful value such as 1 if it is omitted by older client applications.

This approach has the semantic advantage that the same resource is always retrieved from the same URI, but it depends on the code that handles the request to parse the query string and send back the appropriate HTTP response. This approach also suffers from the same complications for implementing HATEOAS as the URI versioning mechanism.

NOTE

Some older web browsers and web proxies will not cache responses for requests that include a query string in the URI. This can degrade performance for web applications that use a web API and that run from within such a web browser.

Header versioning

Rather than appending the version number as a query string parameter, you could implement a custom header that indicates the version of the resource. This approach requires that the client application adds the appropriate header to any requests, although the code handling the client request could use a default value (version 1) if the version header is omitted. The following examples use a custom header named *Custom-Header*. The value of this header indicates the version of web API.

Version 1:

```
GET https://adventure-works.com/customers/3 HTTP/1.1
Custom-Header: api-version=1
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","address":"1 Microsoft Way Redmond WA 98053"}
```

Version 2:

```
GET https://adventure-works.com/customers/3 HTTP/1.1
Custom-Header: api-version=2
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":{"streetAddress":"1 Microsoft Way","city":"Redmond","state":"WA","zipCode":98053}}
```

As with the previous two approaches, implementing HATEOAS requires including the appropriate custom header in any links.

Media type versioning

When a client application sends an HTTP GET request to a web server it should stipulate the format of the content that it can handle by using an `Accept` header, as described earlier in this guidance. Frequently the purpose of the `Accept` header is to allow the client application to specify whether the body of the response should be XML, JSON, or some other common format that the client can parse. However, it is possible to define custom media types that include information enabling the client application to indicate which version of a resource it is expecting.

The following example shows a request that specifies an `Accept` header with the value `application/vnd.adventure-works.v1+json`. The `vnd.adventure-works.v1` element indicates to the web server that it should return version 1 of the resource, while the `json` element specifies that the format of the response body should be JSON:

```
GET https://adventure-works.com/customers/3 HTTP/1.1
Accept: application/vnd.adventure-works.v1+json
```

The code handling the request is responsible for processing the `Accept` header and honoring it as far as possible (the client application may specify multiple formats in the `Accept` header, in which case the web server can choose the most appropriate format for the response body). The web server confirms the format of the data in the response body by using the `Content-Type` header:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.adventure-works.v1+json; charset=utf-8

{"id":3,"name":"Contoso LLC","address":"1 Microsoft Way Redmond WA 98053"}
```

If the `Accept` header does not specify any known media types, the web server could generate an HTTP 406 (Not Acceptable) response message or return a message with a default media type.

This approach is arguably the purest of the versioning mechanisms and lends itself naturally to HATEOAS, which can include the MIME type of related data in resource links.

NOTE

When you select a versioning strategy, you should also consider the implications on performance, especially caching on the web server. The URI versioning and Query String versioning schemes are cache-friendly inasmuch as the same URI/query string combination refers to the same data each time.

The Header versioning and Media Type versioning mechanisms typically require additional logic to examine the values in the custom header or the `Accept` header. In a large-scale environment, many clients using different versions of a web API can result in a significant amount of duplicated data in a server-side cache. This issue can become acute if a client application communicates with a web server through a proxy that implements caching, and that only forwards a request to the web server if it does not currently hold a copy of the requested data in its cache.

Open API Initiative

The [Open API Initiative](#) was created by an industry consortium to standardize REST API descriptions across vendors. As part of this initiative, the Swagger 2.0 specification was renamed the OpenAPI Specification (OAS) and brought under the Open API Initiative.

You may want to adopt OpenAPI for your web APIs. Some points to consider:

- The OpenAPI Specification comes with a set of opinionated guidelines on how a REST API should be designed. That has advantages for interoperability, but requires more care when designing your API to conform to the specification.

- OpenAPI promotes a contract-first approach, rather than an implementation-first approach. Contract-first means you design the API contract (the interface) first and then write code that implements the contract.
- Tools like Swagger can generate client libraries or documentation from API contracts. For example, see [ASP.NET Web API help pages using Swagger](#).

More information

- [Microsoft REST API guidelines](#). Detailed recommendations for designing public REST APIs.
- [Web API checklist](#). A useful list of items to consider when designing and implementing a web API.
- [Open API Initiative](#). Documentation and implementation details on Open API.

Web API implementation

3/10/2022 • 46 minutes to read • [Edit Online](#)

A carefully designed RESTful web API defines the resources, relationships, and navigation schemes that are accessible to client applications. When you implement and deploy a web API, you should consider the physical requirements of the environment hosting the web API and the way in which the web API is constructed rather than the logical structure of the data. This guidance focuses on best practices for implementing a web API and publishing it to make it available to client applications. For detailed information about web API design, see [Web API design](#).

Processing requests

Consider the following points when you implement the code to handle requests.

GET, PUT, DELETE, HEAD, and PATCH actions should be idempotent

The code that implements these requests should not impose any side-effects. The same request repeated over the same resource should result in the same state. For example, sending multiple DELETE requests to the same URI should have the same effect, although the HTTP status code in the response messages may be different. The first DELETE request might return status code 204 (No Content), while a subsequent DELETE request might return status code 404 (Not Found).

NOTE

The article [Idempotency Patterns](#) on Jonathan Oliver's blog provides an overview of idempotency and how it relates to data management operations.

POST actions that create new resources should not have unrelated side-effects

If a POST request is intended to create a new resource, the effects of the request should be limited to the new resource (and possibly any directly related resources if there is some sort of linkage involved). For example, in an e-commerce system, a POST request that creates a new order for a customer might also amend inventory levels and generate billing information, but it should not modify information not directly related to the order or have any other side-effects on the overall state of the system.

Avoid implementing chatty POST, PUT, and DELETE operations

Support POST, PUT and DELETE requests over resource collections. A POST request can contain the details for multiple new resources and add them all to the same collection, a PUT request can replace the entire set of resources in a collection, and a DELETE request can remove an entire collection.

The OData support included in ASP.NET Web API 2 provides the ability to batch requests. A client application can package up several web API requests and send them to the server in a single HTTP request, and receive a single HTTP response that contains the replies to each request. For more information, [Introducing batch support in Web API and Web API OData](#).

Follow the HTTP specification when sending a response

A web API must return messages that contain the correct HTTP status code to enable the client to determine how to handle the result, the appropriate HTTP headers so that the client understands the nature of the result, and a suitably formatted body to enable the client to parse the result.

For example, a POST operation should return status code 201 (Created) and the response message should include the URI of the newly created resource in the Location header of the response message.

Support content negotiation

The body of a response message may contain data in a variety of formats. For example, an HTTP GET request could return data in JSON, or XML format. When the client submits a request, it can include an Accept header that specifies the data formats that it can handle. These formats are specified as media types. For example, a client that issues a GET request that retrieves an image can specify an Accept header that lists the media types that the client can handle, such as `image/jpeg, image/gif, image/png`. When the web API returns the result, it should format the data by using one of these media types and specify the format in the Content-Type header of the response.

If the client does not specify an Accept header, then use a sensible default format for the response body. As an example, the ASP.NET Web API framework defaults to JSON for text-based data.

Provide links to support HATEOAS-style navigation and discovery of resources

The HATEOAS approach enables a client to navigate and discover resources from an initial starting point. This is achieved by using links containing URLs; when a client issues an HTTP GET request to obtain a resource, the response should contain URLs that enable a client application to quickly locate any directly related resources. For example, in a web API that supports an e-commerce solution, a customer may have placed many orders. When a client application retrieves the details for a customer, the response should include links that enable the client application to send HTTP GET requests that can retrieve these orders. Additionally, HATEOAS-style links should describe the other operations (POST, PUT, DELETE, and so on) that each linked resource supports together with the corresponding URI to perform each request. This approach is described in more detail in [API design](#).

Currently there are no standards that govern the implementation of HATEOAS, but the following example illustrates one possible approach. In this example, an HTTP GET request that finds the details for a customer returns a response that includes HATEOAS links that reference the orders for that customer:

```
GET https://adventure-works.com/customers/2 HTTP/1.1
Accept: text/json
...
```

```
HTTP/1.1 200 OK
...
Content-Type: application/json; charset=utf-8
...
Content-Length: ...
{"CustomerID":2,"CustomerName":"Bert","Links": [
    {"rel":"self",
     "href":"https://adventure-works.com/customers/2",
     "action":"GET",
     "types":["text/xml","application/json"]},
    {"rel":"self",
     "href":"https://adventure-works.com/customers/2",
     "action":"PUT",
     "types":["application/x-www-form-urlencoded"]},
    {"rel":"self",
     "href":"https://adventure-works.com/customers/2",
     "action":"DELETE",
     "types":[]},
    {"rel":"orders",
     "href":"https://adventure-works.com/customers/2/orders",
     "action":"GET",
     "types":["text/xml","application/json"]},
    {"rel":"orders",
     "href":"https://adventure-works.com/customers/2/orders",
     "action":"POST",
     "types":["application/x-www-form-urlencoded"]}]
}
```

In this example, the customer data is represented by the `Customer` class shown in the following code snippet. The HATEOAS links are held in the `Links` collection property:

```
public class Customer
{
    public int CustomerID { get; set; }
    public string CustomerName { get; set; }
    public List<Link> Links { get; set; }
    ...
}

public class Link
{
    public string Rel { get; set; }
    public string Href { get; set; }
    public string Action { get; set; }
    public string [] Types { get; set; }
}
```

The HTTP GET operation retrieves the customer data from storage and constructs a `Customer` object, and then populates the `Links` collection. The result is formatted as a JSON response message. Each link comprises the following fields:

- The relationship between the object being returned and the object described by the link. In this case `self` indicates that the link is a reference back to the object itself (similar to a `this` pointer in many object-oriented languages), and `orders` is the name of a collection containing the related order information.
- The hyperlink (`Href`) for the object being described by the link in the form of a URI.
- The type of HTTP request (`Action`) that can be sent to this URI.
- The format of any data (`Types`) that should be provided in the HTTP request or that can be returned in the response, depending on the type of the request.

The HATEOAS links shown in the example HTTP response indicate that a client application can perform the following operations:

- An HTTP GET request to the URI `https://adventure-works.com/customers/2` to fetch the details of the customer (again). The data can be returned as XML or JSON.
- An HTTP PUT request to the URI `https://adventure-works.com/customers/2` to modify the details of the customer. The new data must be provided in the request message in x-www-form-urlencoded format.
- An HTTP DELETE request to the URI `https://adventure-works.com/customers/2` to delete the customer. The request does not expect any additional information or return data in the response message body.
- An HTTP GET request to the URI `https://adventure-works.com/customers/2/orders` to find all the orders for the customer. The data can be returned as XML or JSON.
- An HTTP POST request to the URI `https://adventure-works.com/customers/2/orders` to create a new order for this customer. The data must be provided in the request message in x-www-form-urlencoded format.

Handling exceptions

Consider the following points if an operation throws an uncaught exception.

Capture exceptions and return a meaningful response to clients

The code that implements an HTTP operation should provide comprehensive exception handling rather than letting uncaught exceptions propagate to the framework. If an exception makes it impossible to complete the operation successfully, the exception can be passed back in the response message, but it should include a meaningful description of the error that caused the exception. The exception should also include the appropriate HTTP status code rather than simply returning status code 500 for every situation. For example, if a user request

causes a database update that violates a constraint (such as attempting to delete a customer that has outstanding orders), you should return status code 409 (Conflict) and a message body indicating the reason for the conflict. If some other condition renders the request unachievable, you can return status code 400 (Bad Request). You can find a full list of HTTP status codes on the [Status code definitions](#) page on the W3C website.

The code example traps different conditions and returns an appropriate response.

```
[HttpDelete]
[Route("customers/{id:int}")]
public IHttpActionResult DeleteCustomer(int id)
{
    try
    {
        // Find the customer to be deleted in the repository
        var customerToDelete = repository.GetCustomer(id);

        // If there is no such customer, return an error response
        // with status code 404 (Not Found)
        if (customerToDelete == null)
        {
            return NotFound();
        }

        // Remove the customer from the repository
        // The DeleteCustomer method returns true if the customer
        // was successfully deleted
        if (repository.DeleteCustomer(id))
        {
            // Return a response message with status code 204 (No Content)
            // To indicate that the operation was successful
            return StatusCode(HttpStatusCode.NoContent);
        }
        else
        {
            // Otherwise return a 400 (Bad Request) error response
            return BadRequest(Strings.CustomerNotDeleted);
        }
    }
    catch
    {
        // If an uncaught exception occurs, return an error response
        // with status code 500 (Internal Server Error)
        return InternalServerError();
    }
}
```

TIP

Do not include information that could be useful to an attacker attempting to penetrate your API.

Many web servers trap error conditions themselves before they reach the web API. For example, if you configure authentication for a web site and the user fails to provide the correct authentication information, the web server should respond with status code 401 (Unauthorized). Once a client has been authenticated, your code can perform its own checks to verify that the client should be able access the requested resource. If this authorization fails, you should return status code 403 (Forbidden).

Handle exceptions consistently and log information about errors

To handle exceptions in a consistent manner, consider implementing a global error handling strategy across the entire web API. You should also incorporate error logging which captures the full details of each exception; this error log can contain detailed information as long as it is not made accessible over the web to clients.

Distinguish between client-side errors and server-side errors

The HTTP protocol distinguishes between errors that occur due to the client application (the HTTP 4xx status codes), and errors that are caused by a mishap on the server (the HTTP 5xx status codes). Make sure that you respect this convention in any error response messages.

Optimizing client-side data access

In a distributed environment such as that involving a web server and client applications, one of the primary sources of concern is the network. This can act as a considerable bottleneck, especially if a client application is frequently sending requests or receiving data. Therefore you should aim to minimize the amount of traffic that flows across the network. Consider the following points when you implement the code to retrieve and maintain data:

Support client-side caching

The HTTP 1.1 protocol supports caching in clients and intermediate servers through which a request is routed by the use of the Cache-Control header. When a client application sends an HTTP GET request to the web API, the response can include a Cache-Control header that indicates whether the data in the body of the response can be safely cached by the client or an intermediate server through which the request has been routed, and for how long before it should expire and be considered out-of-date.

The following example shows an HTTP GET request and the corresponding response that includes a Cache-Control header:

```
GET https://adventure-works.com/orders/2 HTTP/1.1
```

```
HTTP/1.1 200 OK
...
Cache-Control: max-age=600, private
Content-Type: text/json; charset=utf-8
Content-Length: ...
>{"orderID":2,"productID":4,"quantity":2,"orderValue":10.00}
```

In this example, the Cache-Control header specifies that the data returned should be expired after 600 seconds, and is only suitable for a single client and must not be stored in a shared cache used by other clients (it is *private*). The Cache-Control header could specify *public* rather than *private* in which case the data can be stored in a shared cache, or it could specify *no-store* in which case the data must **not** be cached by the client. The following code example shows how to construct a Cache-Control header in a response message:

```

public class OrdersController : ApiController
{
    ...
    [Route("api/orders/{id:int:min(0)}")]
    [HttpGet]
    public IHttpActionResult FindOrderByID(int id)
    {
        // Find the matching order
        Order order = ...;
        ...

        // Create a Cache-Control header for the response
        var cacheControlHeader = new CacheControlHeaderValue();
        cacheControlHeader.Private = true;
        cacheControlHeader.MaxAge = new TimeSpan(0, 10, 0);
        ...

        // Return a response message containing the order and the cache control header
        OkResultWithCaching<Order> response = new OkResultWithCaching<Order>(order, this)
        {
            CacheControlHeader = cacheControlHeader
        };
        return response;
    }
    ...
}

```

This code uses a custom `IHttpActionResult` class named `OkResultWithCaching`. This class enables the controller to set the cache header contents:

```

public class OkResultWithCaching<T> : OkNegotiatedContentResult<T>
{
    public OkResultWithCaching(T content, ApiController controller)
        : base(content, controller) { }

    public OkResultWithCaching(T content, IContentNegotiator contentNegotiator, HttpRequestMessage request,
        IEnumerable<MediaTypeFormatter> formatters)
        : base(content, contentNegotiator, request, formatters) { }

    public CacheControlHeaderValue CacheControlHeader { get; set; }
    public EntityTagHeaderValue ETag { get; set; }

    public override async Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)
    {
        HttpResponseMessage response;
        try
        {
            response = await base.ExecuteAsync(cancellationToken);
            response.Headers.CacheControl = this.CacheControlHeader;
            response.Headers.ETag = ETag;
        }
        catch (OperationCanceledException)
        {
            response = new HttpResponseMessage(HttpStatusCode.Conflict) { ReasonPhrase = "Operation was
cancelled" };
        }
        return response;
    }
}

```

NOTE

The HTTP protocol also defines the *no-cache* directive for the Cache-Control header. Rather confusingly, this directive does not mean "do not cache" but rather "revalidate the cached information with the server before returning it"; the data can still be cached, but it is checked each time it is used to ensure that it is still current.

Cache management is the responsibility of the client application or intermediate server, but if properly implemented it can save bandwidth and improve performance by removing the need to fetch data that has already been recently retrieved.

The *max-age* value in the Cache-Control header is only a guide and not a guarantee that the corresponding data won't change during the specified time. The web API should set the max-age to a suitable value depending on the expected volatility of the data. When this period expires, the client should discard the object from the cache.

NOTE

Most modern web browsers support client-side caching by adding the appropriate cache-control headers to requests and examining the headers of the results, as described. However, some older browsers will not cache the values returned from a URL that includes a query string. This is not usually an issue for custom client applications which implement their own cache management strategy based on the protocol discussed here.

Some older proxies exhibit the same behavior and might not cache requests based on URLs with query strings. This could be an issue for custom client applications that connect to a web server through such a proxy.

Provide ETags to optimize query processing

When a client application retrieves an object, the response message can also include an *ETag* (Entity Tag). An ETag is an opaque string that indicates the version of a resource; each time a resource changes the ETag is also modified. This ETag should be cached as part of the data by the client application. The following code example shows how to add an ETag as part of the response to an HTTP GET request. This code uses the `GetHashCode` method of an object to generate a numeric value that identifies the object (you can override this method if necessary and generate your own hash using an algorithm such as MD5) :

```
public class OrdersController : ApiController
{
    ...
    public IHttpActionResult FindOrderByID(int id)
    {
        // Find the matching order
        Order order = ...;
        ...

        var hashedOrder = order.GetHashCode();
        string hashedOrderEtag = $"\"{hashedOrder}\"";
        var eTag = new EntityTagHeaderValue(hashedOrderEtag);

        // Return a response message containing the order and the cache control header
        OkResultWithCaching<Order> response = new OkResultWithCaching<Order>(order, this)
        {
            ...
            ETag = eTag
        };
        return response;
    }
    ...
}
```

The response message posted by the web API looks like this:

```
HTTP/1.1 200 OK
...
Cache-Control: max-age=600, private
Content-Type: text/json; charset=utf-8
ETag: "2147483648"
Content-Length: ...
>{"orderID":2,"productID":4,"quantity":2,"orderValue":10.00}
```

TIP

For security reasons, do not allow sensitive data or data returned over an authenticated (HTTPS) connection to be cached.

A client application can issue a subsequent GET request to retrieve the same resource at any time, and if the resource has changed (it has a different ETag) the cached version should be discarded and the new version added to the cache. If a resource is large and requires a significant amount of bandwidth to transmit back to the client, repeated requests to fetch the same data can become inefficient. To combat this, the HTTP protocol defines the following process for optimizing GET requests that you should support in a web API:

- The client constructs a GET request containing the ETag for the currently cached version of the resource referenced in an If-None-Match HTTP header:

```
GET https://adventure-works.com/orders/2 HTTP/1.1
If-None-Match: "2147483648"
```

- The GET operation in the web API obtains the current ETag for the requested data (order 2 in the above example), and compares it to the value in the If-None-Match header.
- If the current ETag for the requested data matches the ETag provided by the request, the resource has not changed and the web API should return an HTTP response with an empty message body and a status code of 304 (Not Modified).
- If the current ETag for the requested data does not match the ETag provided by the request, then the data has changed and the web API should return an HTTP response with the new data in the message body and a status code of 200 (OK).
- If the requested data no longer exists then the web API should return an HTTP response with the status code of 404 (Not Found).
- The client uses the status code to maintain the cache. If the data has not changed (status code 304) then the object can remain cached and the client application should continue to use this version of the object. If the data has changed (status code 200) then the cached object should be discarded and the new one inserted. If the data is no longer available (status code 404) then the object should be removed from the cache.

NOTE

If the response header contains the Cache-Control header no-store then the object should always be removed from the cache regardless of the HTTP status code.

The code below shows the `FindOrderByID` method extended to support the If-None-Match header. Notice that if the If-None-Match header is omitted, the specified order is always retrieved:

```

public class OrdersController : ApiController
{
    [Route("api/orders/{id:int:min(0)}")]
    [HttpGet]
    public IHttpActionResult FindOrderByID(int id)
    {
        try
        {
            // Find the matching order
            Order order = ...;

            // If there is no such order then return NotFound
            if (order == null)
            {
                return NotFound();
            }

            // Generate the ETag for the order
            var hashedOrder = order.GetHashCode();
            string hashedOrderEtag = $"\"{hashedOrder}\"";

            // Create the Cache-Control and ETag headers for the response
            IHttpActionResult response;
            var cacheControlHeader = new CacheControlHeaderValue();
            cacheControlHeader.Public = true;
            cacheControlHeader.MaxAge = new TimeSpan(0, 10, 0);
            var eTag = new EntityTagHeaderValue(hashedOrderEtag);

            // Retrieve the If-None-Match header from the request (if it exists)
            var nonMatchEtags = Request.Headers.IfNoneMatch;

            // If there is an ETag in the If-None-Match header and
            // this ETag matches that of the order just retrieved,
            // then create a Not Modified response message
            if (nonMatchEtags.Count > 0 &&
                String.CompareOrdinal(nonMatchEtags.First().Tag, hashedOrderEtag) == 0)
            {
                response = new EmptyResultWithCaching()
                {
                    StatusCode = HttpStatusCode.NotModified,
                    CacheControlHeader = cacheControlHeader,
                    ETag = eTag
                };
            }
            // Otherwise create a response message that contains the order details
            else
            {
                response = new OkResultWithCaching<Order>(order, this)
                {
                    CacheControlHeader = cacheControlHeader,
                    ETag = eTag
                };
            }

            return response;
        }
        catch
        {
            return InternalServerError();
        }
    }
    ...
}

```

This example incorporates an additional custom `IHttpActionResult` class named `EmptyResultWithCaching`. This class simply acts as a wrapper around an `HttpResponseMessage` object that does not contain a response body:

```

public class EmptyResultWithCaching : IHttpActionResult
{
    public CacheControlHeaderValue CacheControlHeader { get; set; }
    public EntityTagHeaderValue ETag { get; set; }
    public HttpStatusCode StatusCode { get; set; }
    public Uri Location { get; set; }

    public async Task<HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken)
    {
        HttpResponseMessage response = new HttpResponseMessage(HttpStatusCode.OK);
        response.Headers.CacheControl = this.CacheControlHeader;
        response.Headers.ETag = this.ETag;
        response.Headers.Location = this.Location;
        return response;
    }
}

```

TIP

In this example, the ETag for the data is generated by hashing the data retrieved from the underlying data source. If the ETag can be computed in some other way, then the process can be optimized further and the data only needs to be fetched from the data source if it has changed. This approach is especially useful if the data is large or accessing the data source can result in significant latency (for example, if the data source is a remote database).

Use ETags to Support Optimistic Concurrency

To enable updates over previously cached data, the HTTP protocol supports an optimistic concurrency strategy. If, after fetching and caching a resource, the client application subsequently sends a PUT or DELETE request to change or remove the resource, it should include in If-Match header that references the ETag. The web API can then use this information to determine whether the resource has already been changed by another user since it was retrieved and send an appropriate response back to the client application as follows:

- The client constructs a PUT request containing the new details for the resource and the ETag for the currently cached version of the resource referenced in an If-Match HTTP header. The following example shows a PUT request that updates an order:

```

PUT https://adventure-works.com/orders/1 HTTP/1.1
If-Match: "2282343857"
Content-Type: application/x-www-form-urlencoded
Content-Length: ...
productID=3&quantity=5&orderValue=250

```

- The PUT operation in the web API obtains the current ETag for the requested data (order 1 in the above example), and compares it to the value in the If-Match header.
- If the current ETag for the requested data matches the ETag provided by the request, the resource has not changed and the web API should perform the update, returning a message with HTTP status code 204 (No Content) if it is successful. The response can include Cache-Control and ETag headers for the updated version of the resource. The response should always include the Location header that references the URI of the newly updated resource.
- If the current ETag for the requested data does not match the ETag provided by the request, then the data has been changed by another user since it was fetched and the web API should return an HTTP response with an empty message body and a status code of 412 (Precondition Failed).
- If the resource to be updated no longer exists then the web API should return an HTTP response with the status code of 404 (Not Found).

- The client uses the status code and response headers to maintain the cache. If the data has been updated (status code 204) then the object can remain cached (as long as the Cache-Control header does not specify no-store) but the ETag should be updated. If the data was changed by another user changed (status code 412) or not found (status code 404) then the cached object should be discarded.

The next code example shows an implementation of the PUT operation for the Orders controller:

```

public class OrdersController : ApiController
{
    [HttpPut]
    [Route("api/orders/{id:int}")]
    public IHttpActionResult UpdateExistingOrder(int id, DTOOrder order)
    {
        try
        {
            var baseUri = Constants.GetUriFromConfig();
            var orderToUpdate = this.ordersRepository.GetOrder(id);
            if (orderToUpdate == null)
            {
                return NotFound();
            }

            var hashedOrder = orderToUpdate.GetHashCode();
            string hashedOrderEtag = $"\"{hashedOrder}\"";

            // Retrieve the If-Match header from the request (if it exists)
            var matchEtags = Request.Headers.IfMatch;

            // If there is an ETag in the If-Match header and
            // this ETag matches that of the order just retrieved,
            // or if there is no ETag, then update the Order
            if (((matchEtags.Count > 0 &&
                  String.CompareOrdinal(matchEtags.First().Tag, hashedOrderEtag) == 0)) ||
                matchEtags.Count == 0)
            {
                // Modify the order
                orderToUpdate.OrderValue = order.OrderValue;
                orderToUpdate.ProductID = order.ProductID;
                orderToUpdate.Quantity = order.Quantity;

                // Save the order back to the data store
                // ...

                // Create the No Content response with Cache-Control, ETag, and Location headers
                var cacheControlHeader = new CacheControlHeaderValue();
                cacheControlHeader.Private = true;
                cacheControlHeader.MaxAge = new TimeSpan(0, 10, 0);

                hashedOrder = order.GetHashCode();
                hashedOrderEtag = $"\"{hashedOrder}\"";
                var eTag = new EntityTagHeaderValue(hashedOrderEtag);

                var location = new Uri($"{baseUri}/{Constants.ORDERS}/{id}");
                var response = new EmptyResultWithCaching()
                {
                    StatusCode = HttpStatusCode.NoContent,
                    CacheControlHeader = cacheControlHeader,
                    ETag = eTag,
                    Location = location
                };
            }

            return response;
        }

        // Otherwise return a Precondition Failed response
        return StatusCode(HttpStatusCode.PreconditionFailed);
    }
}

```

```
        ,
        catch
        {
            return InternalServerError();
        }
    }
    ...
}
```

TIP

Use of the If-Match header is entirely optional, and if it is omitted the web API will always attempt to update the specified order, possibly blindly overwriting an update made by another user. To avoid problems due to lost updates, always provide an If-Match header.

Handling large requests and responses

There may be occasions when a client application needs to issue requests that send or receive data that may be several megabytes (or bigger) in size. Waiting while this amount of data is transmitted could cause the client application to become unresponsive. Consider the following points when you need to handle requests that include significant amounts of data:

Optimize requests and responses that involve large objects

Some resources may be large objects or include large fields, such as graphics images or other types of binary data. A web API should support streaming to enable optimized uploading and downloading of these resources.

The HTTP protocol provides the chunked transfer encoding mechanism to stream large data objects back to a client. When the client sends an HTTP GET request for a large object, the web API can send the reply back in piecemeal *chunks* over an HTTP connection. The length of the data in the reply may not be known initially (it might be generated), so the server hosting the web API should send a response message with each chunk that specifies the Transfer-Encoding: Chunked header rather than a Content-Length header. The client application can receive each chunk in turn to build up the complete response. The data transfer completes when the server sends back a final chunk with zero size.

A single request could conceivably result in a massive object that consumes considerable resources. If during the streaming process the web API determines that the amount of data in a request has exceeded some acceptable bounds, it can abort the operation and return a response message with status code 413 (Request Entity Too Large).

You can minimize the size of large objects transmitted over the network by using HTTP compression. This approach helps to reduce the amount of network traffic and the associated network latency, but at the cost of requiring additional processing at the client and the server hosting the web API. For example, a client application that expects to receive compressed data can include an Accept-Encoding: gzip request header (other data compression algorithms can also be specified). If the server supports compression it should respond with the content held in gzip format in the message body and the Content-Encoding: gzip response header.

You can combine encoded compression with streaming; compress the data first before streaming it, and specify the gzip content encoding and chunked transfer encoding in the message headers. Also note that some web servers (such as Internet Information Server) can be configured to automatically compress HTTP responses regardless of whether the web API compresses the data or not.

Implement partial responses for clients that do not support asynchronous operations

As an alternative to asynchronous streaming, a client application can explicitly request data for large objects in chunks, known as partial responses. The client application sends an HTTP HEAD request to obtain information about the object. If the web API supports partial responses it should respond to the HEAD request with a response message that contains an Accept-Ranges header and a Content-Length header that indicates the total

size of the object, but the body of the message should be empty. The client application can use this information to construct a series of GET requests that specify a range of bytes to receive. The web API should return a response message with HTTP status 206 (Partial Content), a Content-Length header that specifies the actual amount of data included in the body of the response message, and a Content-Range header that indicates which part (such as bytes 4000 to 8000) of the object this data represents.

HTTP HEAD requests and partial responses are described in more detail in [API design](#).

Avoid sending unnecessary 100-Continue status messages in client applications

A client application that is about to send a large amount of data to a server may determine first whether the server is actually willing to accept the request. Prior to sending the data, the client application can submit an HTTP request with an Expect: 100-Continue header, a Content-Length header that indicates the size of the data, but an empty message body. If the server is willing to handle the request, it should respond with a message that specifies the HTTP status 100 (Continue). The client application can then proceed and send the complete request including the data in the message body.

If you are hosting a service by using IIS, the HTTP.sys driver automatically detects and handles Expect: 100-Continue headers before passing requests to your web application. This means that you are unlikely to see these headers in your application code, and you can assume that IIS has already filtered any messages that it deems to be unfit or too large.

If you are building client applications by using the .NET Framework, then all POST and PUT messages will first send messages with Expect: 100-Continue headers by default. As with the server-side, the process is handled transparently by the .NET Framework. However, this process results in each POST and PUT request causing two round-trips to the server, even for small requests. If your application is not sending requests with large amounts of data, you can disable this feature by using the `ServicePointManager` class to create `ServicePoint` objects in the client application. A `ServicePoint` object handles the connections that the client makes to a server based on the scheme and host fragments of URIs that identify resources on the server. You can then set the `Expect100Continue` property of the `ServicePoint` object to false. All subsequent POST and PUT requests made by the client through a URI that matches the scheme and host fragments of the `ServicePoint` object will be sent without Expect: 100-Continue headers. The following code shows how to configure a `ServicePoint` object that configures all requests sent to URIs with a scheme of `http` and a host of `www.contoso.com`.

```
Uri uri = new Uri("https://www.contoso.com/");
ServicePoint sp = ServicePointManager.FindServicePoint(uri);
sp.Expect100Continue = false;
```

You can also set the static `Expect100Continue` property of the `ServicePointManager` class to specify the default value of this property for all subsequently created `ServicePoint` objects.

Support pagination for requests that may return large numbers of objects

If a collection contains a large number of resources, issuing a GET request to the corresponding URI could result in significant processing on the server hosting the web API affecting performance, and generate a significant amount of network traffic resulting in increased latency.

To handle these cases, the web API should support query strings that enable the client application to refine requests or fetch data in more manageable, discrete blocks (or pages). The code below shows the `GetAllOrders` method in the `orders` controller. This method retrieves the details of orders. If this method was unconstrained, it could conceivably return a large amount of data. The `limit` and `offset` parameters are intended to reduce the volume of data to a smaller subset, in this case only the first 10 orders by default:

```
public class OrdersController : ApiController
{
    ...
    [Route("api/orders")]
    [HttpGet]
    public IEnumerable<Order> GetAllOrders(int limit=10, int offset=0)
    {
        // Find the number of orders specified by the limit parameter
        // starting with the order specified by the offset parameter
        var orders = ...
        return orders;
    }
    ...
}
```

A client application can issue a request to retrieve 30 orders starting at offset 50 by using the URI

<https://www.adventure-works.com/api/orders?limit=30&offset=50>.

TIP

Avoid enabling client applications to specify query strings that result in a URI that is more than 2000 characters long. Many web clients and servers cannot handle URIs that are this long.

Maintaining responsiveness, scalability, and availability

The same web API might be used by many client applications running anywhere in the world. It is important to ensure that the web API is implemented to maintain responsiveness under a heavy load, to be scalable to support a highly varying workload, and to guarantee availability for clients that perform business-critical operations. Consider the following points when determining how to meet these requirements:

Provide asynchronous support for long-running requests

A request that might take a long time to process should be performed without blocking the client that submitted the request. The web API can perform some initial checking to validate the request, initiate a separate task to perform the work, and then return a response message with HTTP code 202 (Accepted). The task could run asynchronously as part of the web API processing, or it could be offloaded to a background task.

The web API should also provide a mechanism to return the results of the processing to the client application. You can achieve this by providing a polling mechanism for client applications to periodically query whether the processing has finished and obtain the result, or enabling the web API to send a notification when the operation has completed.

You can implement a simple polling mechanism by providing a *polling* URI that acts as a virtual resource using the following approach:

1. The client application sends the initial request to the web API.
2. The web API stores information about the request in a table held in table storage or Microsoft Azure Cache, and generates a unique key for this entry, possibly in the form of a GUID.
3. The web API initiates the processing as a separate task. The web API records the state of the task in the table as *Running*.
4. The web API returns a response message with HTTP status code 202 (Accepted), and the GUID of the table entry in the body of the message.
5. When the task has completed, the web API stores the results in the table, and sets the state of the task to *Complete*. Note that if the task fails, the web API could also store information about the failure and set the status to *Failed*.
6. While the task is running, the client can continue performing its own processing. It can periodically send a

request to the URI `/polling/{guid}` where `{guid}` is the GUID returned in the 202 response message by the web API.

7. The web API at the `/polling/{guid}` URI queries the state of the corresponding task in the table and returns a response message with HTTP status code 200 (OK) containing this state (*Running*, *Complete*, or *Failed*). If the task has completed or failed, the response message can also include the results of the processing or any information available about the reason for the failure.

Options for implementing notifications include:

- Using a notification hub to push asynchronous responses to client applications. For more information, see [Send notifications to specific users by using Azure Notification Hubs](#).
- Using the Comet model to retain a persistent network connection between the client and the server hosting the web API, and using this connection to push messages from the server back to the client. The MSDN magazine article [Building a Simple Comet Application in the Microsoft .NET Framework](#) describes an example solution.
- Using SignalR to push data in real time from the web server to the client over a persistent network connection. SignalR is available for ASP.NET web applications as a NuGet package. You can find more information on the [ASP.NET SignalR](#) website.

Ensure that each request is stateless

Each request should be considered atomic. There should be no dependencies between one request made by a client application and any subsequent requests submitted by the same client. This approach assists in scalability; instances of the web service can be deployed on a number of servers. Client requests can be directed at any of these instances and the results should always be the same. It also improves availability for a similar reason; if a web server fails requests can be routed to another instance (by using Azure Traffic Manager) while the server is restarted with no ill effects on client applications.

Track clients and implement throttling to reduce the chances of DOS attacks

If a specific client makes a large number of requests within a given period of time it might monopolize the service and affect the performance of other clients. To mitigate this issue, a web API can monitor calls from client applications either by tracking the IP address of all incoming requests or by logging each authenticated access. You can use this information to limit resource access. If a client exceeds a defined limit, the web API can return a response message with status 503 (Service Unavailable) and include a `Retry-After` header that specifies when the client can send the next request without it being declined. This strategy can help to reduce the chances of a Denial Of Service (DOS) attack from a set of clients stalling the system.

Manage persistent HTTP connections carefully

The HTTP protocol supports persistent HTTP connections where they are available. The HTTP 1.0 specification added the `Connection:Keep-Alive` header that enables a client application to indicate to the server that it can use the same connection to send subsequent requests rather than opening new ones. The connection closes automatically if the client does not reuse the connection within a period defined by the host. This behavior is the default in HTTP 1.1 as used by Azure services, so there is no need to include `Keep-Alive` headers in messages.

Keeping a connection open can help to improve responsiveness by reducing latency and network congestion, but it can be detrimental to scalability by keeping unnecessary connections open for longer than required, limiting the ability of other concurrent clients to connect. It can also affect battery life if the client application is running on a mobile device; if the application only makes occasional requests to the server, maintaining an open connection can cause the battery to drain more quickly. To ensure that a connection is not made persistent with HTTP 1.1, the client can include a `Connection:Close` header with messages to override the default behavior. Similarly, if a server is handling a very large number of clients it can include a `Connection:Close` header in response messages which should close the connection and save server resources.

NOTE

Persistent HTTP connections are a purely optional feature to reduce the network overhead associated with repeatedly establishing a communications channel. Neither the web API nor the client application should depend on a persistent HTTP connection being available. Do not use persistent HTTP connections to implement Comet-style notification systems; instead you should use sockets (or web sockets if available) at the TCP layer. Finally, note Keep-Alive headers are of limited use if a client application communicates with a server via a proxy; only the connection with the client and the proxy will be persistent.

Publishing and managing a web API

To make a web API available for client applications, the web API must be deployed to a host environment. This environment is typically a web server, although it may be some other type of host process. You should consider the following points when publishing a web API:

- All requests must be authenticated and authorized, and the appropriate level of access control must be enforced.
- A commercial web API might be subject to various quality guarantees concerning response times. It is important to ensure that host environment is scalable if the load can vary significantly over time.
- It may be necessary to meter requests for monetization purposes.
- It might be necessary to regulate the flow of traffic to the web API, and implement throttling for specific clients that have exhausted their quotas.
- Regulatory requirements might mandate logging and auditing of all requests and responses.
- To ensure availability, it may be necessary to monitor the health of the server hosting the web API and restart it if necessary.

It is useful to be able to decouple these issues from the technical issues concerning the implementation of the web API. For this reason, consider creating a [façade](#), running as a separate process and that routes requests to the web API. The façade can provide the management operations and forward validated requests to the web API. Using a façade can also bring many functional advantages, including:

- Acting as an integration point for multiple web APIs.
- Transforming messages and translating communications protocols for clients built by using varying technologies.
- Caching requests and responses to reduce load on the server hosting the web API.

Testing a web API

A web API should be tested as thoroughly as any other piece of software. You should consider creating unit tests to validate the functionality.

The nature of a web API brings its own additional requirements to verify that it operates correctly. You should pay particular attention to the following aspects:

- Test all routes to verify that they invoke the correct operations. Be especially aware of HTTP status code 405 (Method Not Allowed) being returned unexpectedly as this can indicate a mismatch between a route and the HTTP methods (GET, POST, PUT, DELETE) that can be dispatched to that route.

Send HTTP requests to routes that do not support them, such as submitting a POST request to a specific resource (POST requests should only be sent to resource collections). In these cases, the only valid response *should* be status code 405 (Not Allowed).

- Verify that all routes are protected properly and are subject to the appropriate authentication and authorization checks.

NOTE

Some aspects of security such as user authentication are most likely to be the responsibility of the host environment rather than the web API, but it is still necessary to include security tests as part of the deployment process.

- Test the exception handling performed by each operation and verify that an appropriate and meaningful HTTP response is passed back to the client application.
- Verify that request and response messages are well-formed. For example, if an HTTP POST request contains the data for a new resource in x-www-form-urlencoded format, confirm that the corresponding operation correctly parses the data, creates the resources, and returns a response containing the details of the new resource, including the correct Location header.
- Verify all links and URIs in response messages. For example, an HTTP POST message should return the URI of the newly created resource. All HATEOAS links should be valid.
- Ensure that each operation returns the correct status codes for different combinations of input. For example:
 - If a query is successful, it should return status code 200 (OK)
 - If a resource is not found, the operation should return HTTP status code 404 (Not Found).
 - If the client sends a request that successfully deletes a resource, the status code should be 204 (No Content).
 - If the client sends a request that creates a new resource, the status code should be 201 (Created).

Watch out for unexpected response status codes in the 5xx range. These messages are usually reported by the host server to indicate that it was unable to fulfill a valid request.

- Test the different request header combinations that a client application can specify and ensure that the web API returns the expected information in response messages.
- Test query strings. If an operation can take optional parameters (such as pagination requests), test the different combinations and order of parameters.
- Verify that asynchronous operations complete successfully. If the web API supports streaming for requests that return large binary objects (such as video or audio), ensure that client requests are not blocked while the data is streamed. If the web API implements polling for long-running data modification operations, verify that the operations report their status correctly as they proceed.

You should also create and run performance tests to check that the web API operates satisfactorily under duress. You can build a web performance and load test project by using Visual Studio Ultimate.

Using Azure API Management

On Azure, consider using [Azure API Management](#) to publish and manage a web API. Using this facility, you can generate a service that acts as a façade for one or more web APIs. The service is itself a scalable web service that you can create and configure by using the Azure portal. You can use this service to publish and manage a web API as follows:

1. Deploy the web API to a website, Azure cloud service, or Azure virtual machine.
2. Connect the API management service to the web API. Requests sent to the URL of the management API are mapped to URIs in the web API. The same API management service can route requests to more than one web API. This enables you to aggregate multiple web APIs into a single management service. Similarly, the same web API can be referenced from more than one API management service if you need

to restrict or partition the functionality available to different applications.

NOTE

The URIs in HATEOAS links generated as part of the response for HTTP GET requests should reference the URL of the API management service and not the web server hosting the web API.

3. For each web API, specify the HTTP operations that the web API exposes together with any optional parameters that an operation can take as input. You can also configure whether the API management service should cache the response received from the web API to optimize repeated requests for the same data. Record the details of the HTTP responses that each operation can generate. This information is used to generate documentation for developers, so it is important that it is accurate and complete.

You can either define operations manually using the wizards provided by the Azure portal, or you can import them from a file containing the definitions in WADL or Swagger format.

4. Configure the security settings for communications between the API management service and the web server hosting the web API. The API management service currently supports Basic authentication and mutual authentication using certificates, and OAuth 2.0 user authorization.
5. Create a product. A product is the unit of publication; you add the web APIs that you previously connected to the management service to the product. When the product is published, the web APIs become available to developers.

NOTE

Prior to publishing a product, you can also define user-groups that can access the product and add users to these groups. This gives you control over the developers and applications that can use the web API. If a web API is subject to approval, prior to being able to access it a developer must send a request to the product administrator. The administrator can grant or deny access to the developer. Existing developers can also be blocked if circumstances change.

6. Configure policies for each web API. Policies govern aspects such as whether cross-domain calls should be allowed, how to authenticate clients, whether to convert between XML and JSON data formats transparently, whether to restrict calls from a given IP range, usage quotas, and whether to limit the call rate. Policies can be applied globally across the entire product, for a single web API in a product, or for individual operations in a web API.

For more information, see the [API Management documentation](#).

TIP

Azure provides the Azure Traffic Manager which enables you to implement failover and load-balancing, and reduce latency across multiple instances of a web site hosted in different geographic locations. You can use Azure Traffic Manager in conjunction with the API Management Service; the API Management Service can route requests to instances of a web site through Azure Traffic Manager. For more information, see [Traffic Manager routing methods](#).

In this structure, if you are using custom DNS names for your web sites, you should configure the appropriate CNAME record for each web site to point to the DNS name of the Azure Traffic Manager web site.

Supporting client-side developers

Developers constructing client applications typically require information on how to access the web API, and documentation concerning the parameters, data types, return types, and return codes that describe the different

requests and responses between the web service and the client application.

Document the REST operations for a web API

The Azure API Management Service includes a developer portal that describes the REST operations exposed by a web API. When a product has been published it appears on this portal. Developers can use this portal to sign up for access; the administrator can then approve or deny the request. If the developer is approved, they are assigned a subscription key that is used to authenticate calls from the client applications that they develop. This key must be provided with each web API call otherwise it will be rejected.

This portal also provides:

- Documentation for the product, listing the operations that it exposes, the parameters required, and the different responses that can be returned. Note that this information is generated from the details provided in step 3 in the list in the Publishing a web API by using the Microsoft Azure API Management Service section.
- Code snippets that show how to invoke operations from several languages, including JavaScript, C#, Java, Ruby, Python, and PHP.
- A developers' console that enables a developer to send an HTTP request to test each operation in the product and view the results.
- A page where the developer can report any issues or problems found.

The Azure portal enables you to customize the developer portal to change the styling and layout to match the branding of your organization.

Implement a client SDK

Building a client application that invokes REST requests to access a web API requires writing a significant amount of code to construct each request and format it appropriately, send the request to the server hosting the web service, and parse the response to work out whether the request succeeded or failed and extract any data returned. To insulate the client application from these concerns, you can provide an SDK that wraps the REST interface and abstracts these low-level details inside a more functional set of methods. A client application uses these methods, which transparently convert calls into REST requests and then convert the responses back into method return values. This is a common technique that is implemented by many services, including the Azure SDK.

Creating a client-side SDK is a considerable undertaking as it has to be implemented consistently and tested carefully. However, much of this process can be made mechanical, and many vendors supply tools that can automate many of these tasks.

Monitoring a web API

Depending on how you have published and deployed your web API you can monitor the web API directly, or you can gather usage and health information by analyzing the traffic that passes through the API Management service.

Monitoring a web API directly

If you have implemented your web API by using the ASP.NET Web API template (either as a Web API project or as a Web role in an Azure cloud service) and Visual Studio 2013, you can gather availability, performance, and usage data by using ASP.NET Application Insights. Application Insights is a package that transparently tracks and records information about requests and responses when the web API is deployed to the cloud; once the package is installed and configured, you don't need to amend any code in your web API to use it. When you deploy the web API to an Azure web site, all traffic is examined and the following statistics are gathered:

- Server response time.
- Number of server requests and the details of each request.
- The top slowest requests in terms of average response time.

- The details of any failed requests.
- The number of sessions initiated by different browsers and user agents.
- The most frequently viewed pages (primarily useful for web applications rather than web APIs).
- The different user roles accessing the web API.

You can view this data in real time in the Azure portal. You can also create web tests that monitor the health of the web API. A web test sends a periodic request to a specified URI in the web API and captures the response. You can specify the definition of a successful response (such as HTTP status code 200), and if the request does not return this response you can arrange for an alert to be sent to an administrator. If necessary, the administrator can restart the server hosting the web API if it has failed.

For more information, see [Application Insights - Get started with ASP.NET](#).

Monitoring a web API through the API Management Service

If you have published your web API by using the API Management service, the API Management page on the Azure portal contains a dashboard that enables you to view the overall performance of the service. The Analytics page enables you to drill down into the details of how the product is being used. This page contains the following tabs:

- **Usage.** This tab provides information about the number of API calls made and the bandwidth used to handle these calls over time. You can filter usage details by product, API, and operation.
- **Health.** This tab enables you to view the outcome of API requests (the HTTP status codes returned), the effectiveness of the caching policy, the API response time, and the service response time. Again, you can filter health data by product, API, and operation.
- **Activity.** This tab provides a text summary of the numbers of successful calls, failed calls, blocked calls, average response time, and response times for each product, web API, and operation. This page also lists the number of calls made by each developer.
- **At a glance.** This tab displays a summary of the performance data, including the developers responsible for making the most API calls, and the products, web APIs, and operations that received these calls.

You can use this information to determine whether a particular web API or operation is causing a bottleneck, and if necessary scale the host environment and add more servers. You can also ascertain whether one or more applications are using a disproportionate volume of resources and apply the appropriate policies to set quotas and limit call rates.

NOTE

You can change the details for a published product, and the changes are applied immediately. For example, you can add or remove an operation from a web API without requiring that you republish the product that contains the web API.

More information

- [ASP.NET Web API OData](#) contains examples and further information on implementing an OData web API by using ASP.NET.
- [Introducing batch support in Web API and Web API OData](#) describes how to implement batch operations in a web API by using OData.
- [Idempotency patterns](#) on Jonathan Oliver's blog provides an overview of idempotency and how it relates to data management operations.
- [Status code definitions](#) on the W3C website contains a full list of HTTP status codes and their descriptions.
- [Run background tasks with WebJobs](#) provides information and examples on using WebJobs to perform background operations.
- [Azure Notification Hubs notify users](#) shows how to use an Azure Notification Hub to push asynchronous

responses to client applications.

- [API Management](#) describes how to publish a product that provides controlled and secure access to a web API.
- [Azure API Management REST API reference](#) describes how to use the API Management REST API to build custom management applications.
- [Traffic Manager routing methods](#) summarizes how Azure Traffic Manager can be used to load-balance requests across multiple instances of a website hosting a web API.
- [Application Insights - Get started with ASP.NET](#) provides detailed information on installing and configuring Application Insights in an ASP.NET Web API project.

Autoscaling

3/10/2022 • 15 minutes to read • [Edit Online](#)

Autoscaling is the process of dynamically allocating resources to match performance requirements. As the volume of work grows, an application may need additional resources to maintain the desired performance levels and satisfy service-level agreements (SLAs). As demand slackens and the additional resources are no longer needed, they can be de-allocated to minimize costs.

Autoscaling takes advantage of the elasticity of cloud-hosted environments while easing management overhead. It reduces the need for an operator to continually monitor the performance of a system and make decisions about adding or removing resources.

There are two main ways that an application can scale:

- **Vertical scaling**, also called scaling up and down, means changing the capacity of a resource. For example, you could move an application to a larger VM size. Vertical scaling often requires making the system temporarily unavailable while it is being redeployed. Therefore, it's less common to automate vertical scaling.
- **Horizontal scaling**, also called scaling out and in, means adding or removing instances of a resource. The application continues running without interruption as new resources are provisioned. When the provisioning process is complete, the solution is deployed on these additional resources. If demand drops, the additional resources can be shut down cleanly and deallocated.

Many cloud-based systems, including Microsoft Azure, support automatic horizontal scaling. The rest of this article focuses on horizontal scaling.

NOTE

Autoscaling mostly applies to compute resources. While it's possible to horizontally scale a database or message queue, this usually involves [data partitioning](#), which is generally not automated.

Autoscaling components

An autoscaling strategy typically involves the following pieces:

- Instrumentation and monitoring systems at the application, service, and infrastructure levels. These systems capture key metrics, such as response times, queue lengths, CPU utilization, and memory usage.
- Decision-making logic that evaluates these metrics against predefined thresholds or schedules, and decides whether to scale.
- Components that scale the system.
- Testing, monitoring, and tuning of the autoscaling strategy to ensure that it functions as expected.

Azure provides built-in autoscaling mechanisms that address common scenarios. If a particular service or technology does not have built-in autoscaling functionality, or if you have specific autoscaling requirements beyond its capabilities, you might consider a custom implementation. A custom implementation would collect operational and system metrics, analyze the metrics, and then scale resources accordingly.

Configure autoscaling for an Azure solution

Azure provides built-in autoscaling for most compute options.

- **Azure Virtual Machines** autoscale via [virtual machine scale sets](#), which manage a set of Azure virtual machines as a group. See [How to use automatic scaling and virtual machine scale sets](#).
- **Service Fabric** also supports autoscaling through virtual machine scale sets. Every node type in a Service Fabric cluster is set up as a separate virtual machine scale set. That way, each node type can be scaled in or out independently. See [Scale a Service Fabric cluster in or out using autoscale rules](#).
- **Azure App Service** has built-in autoscaling. Autoscale settings apply to all of the apps within an App Service. See [Scale instance count manually or automatically](#).
- **Azure Cloud Services** has built-in autoscaling at the role level. See [How to configure auto scaling for a Cloud Service in the portal](#).

These compute options all use [Azure Monitor autoscale](#) to provide a common set of autoscaling functionality.

- **Azure Functions** differs from the previous compute options, because you don't need to configure any autoscale rules. Instead, Azure Functions automatically allocates compute power when your code is running, scaling out as necessary to handle load. For more information, see [Choose the correct hosting plan for Azure Functions](#).

Finally, a custom autoscaling solution can sometimes be useful. For example, you could use Azure diagnostics and application-based metrics, along with custom code to monitor and export the application metrics. Then you could define custom rules based on these metrics, and use Resource Manager REST APIs to trigger autoscaling. However, a custom solution is not simple to implement, and should be considered only if none of the previous approaches can fulfill your requirements.

Use the built-in autoscaling features of the platform, if they meet your requirements. If not, carefully consider whether you really need more complex scaling features. Examples of additional requirements may include more granularity of control, different ways to detect trigger events for scaling, scaling across subscriptions, and scaling other types of resources.

Use Azure Monitor autoscale

[Azure Monitor autoscale](#) provide a common set of autoscaling functionality for virtual machine scale sets, Azure App Service, and Azure Cloud Service. Scaling can be performed on a schedule, or based on a runtime metric, such as CPU or memory usage.

Examples:

- Scale out to 10 instances on weekdays, and scale in to 4 instances on Saturday and Sunday.
- Scale out by one instance if average CPU usage is above 70%, and scale in by one instance if CPU usage falls below 50%.
- Scale out by one instance if the number of messages in a queue exceeds a certain threshold.

Scale up the resource when load increases to ensure availability. Similarly, at times of low usage, scale down, so you can optimize cost. Always use a scale-out and scale-in rule combination. Otherwise, the autoscaling takes place only in one direction until it reaches the threshold (maximum or minimum instance counts) set in the profile.

Select a default instance count that's safe for your workload. It's scaled based on that value if maximum or minimum instance counts are not set.

For a list of built-in metrics, see [Azure Monitor autoscaling common metrics](#). You can also implement custom metrics by using Application Insights.

You can configure autoscaling by using PowerShell, the Azure CLI, an Azure Resource Manager template, or the Azure portal. For more detailed control, use the [Azure Resource Manager REST API](#). The [Azure Monitoring](#)

[Service Management Library](#) and the [Microsoft Insights Library](#) (in preview) are SDKs that allow collecting metrics from different resources, and perform autoscaling by making use of the REST APIs. For resources where Azure Resource Manager support isn't available, or if you are using Azure Cloud Services, the Service Management REST API can be used for autoscaling. In all other cases, use Azure Resource Manager.

Consider the following points when using Azure autoscale:

- Consider whether you can predict the load on the application accurately enough to use scheduled autoscaling, adding and removing instances to meet anticipated peaks in demand. If this isn't possible, use reactive autoscaling based on runtime metrics, in order to handle unpredictable changes in demand. Typically, you can combine these approaches. For example, create a strategy that adds resources based on a schedule of the times when you know the application is busiest. This helps to ensure that capacity is available when required, without any delay from starting new instances. For each scheduled rule, define metrics that allow reactive autoscaling during that period to ensure that the application can handle sustained but unpredictable peaks in demand.
- It's often difficult to understand the relationship between metrics and capacity requirements, especially when an application is initially deployed. Provision a little extra capacity at the beginning, and then monitor and tune the autoscaling rules to bring the capacity closer to the actual load.
- Configure the autoscaling rules, and then monitor the performance of your application over time. Use the results of this monitoring to adjust the way in which the system scales if necessary. However, keep in mind that autoscaling is not an instantaneous process. It takes time to react to a metric such as average CPU utilization exceeding (or falling below) a specified threshold.
- Autoscaling rules that use a detection mechanism based on a measured trigger attribute (such as CPU usage or queue length) use an aggregated value over time, rather than instantaneous values, to trigger an autoscaling action. By default, the aggregate is an average of the values. This prevents the system from reacting too quickly, or causing rapid oscillation. It also allows time for new instances that are automatically started to settle into running mode, preventing additional autoscaling actions from occurring while the new instances are starting up. For Azure Cloud Services and Azure Virtual Machines, the default period for the aggregation is 45 minutes, so it can take up to this period of time for the metric to trigger autoscaling in response to spikes in demand. You can change the aggregation period by using the SDK, but periods of less than 25 minutes may cause unpredictable results. For Web Apps, the averaging period is much shorter, allowing new instances to be available in about five minutes after a change to the average trigger measure.
- Avoid *flapping* where scale-in and scale-out actions continually go back and forth. Suppose there are two instances, and upper limit is 80% CPU, lower limit is 60%. When the load is at 85%, another instance is added. After some time, the load decreases to 60%. Before scaling in, the autoscale service calculates the distribution of total load (of three instances) when an instance is removed, taking it to 90%. This means it would have to scale out again immediately. So, it skips scaling-in and you might never see the expected scaling results.

The flapping situation can be controlled by choosing an adequate margin between the scale-out and scale-in thresholds.

- Manual scaling is reset by maximum and minimum number of instances used for autoscaling. If you manually update the instance count to a value higher or lower than the maximum value, the autoscale engine automatically scales back to the minimum (if lower) or the maximum (if higher). For example, you set the range between 3 and 6. If you have one running instance, the autoscale engine scales to three instances on its next run. Likewise, if you manually set the scale to eight instances, on the next run autoscale will scale it back to six instances on its next run. Manual scaling is temporary unless you reset the autoscale rules as well.
- The autoscale engine processes only one profile at a time. If a condition is not met, then it checks for the

next profile. Keep key metrics out of the default profile because that profile is checked last. Within a profile, you can have multiple rules. On scale-out, autoscale runs if any rule is met. On scale-in, autoscale require all rules to be met.

For details about how Azure Monitor scales, see [Best practices for Autoscale](#).

- If you configure autoscaling using the SDK rather than the portal, you can specify a more detailed schedule during which the rules are active. You can also create your own metrics and use them with or without any of the existing ones in your autoscaling rules. For example, you may wish to use alternative counters, such as the number of requests per second or the average memory availability, or use custom counters to measure specific business processes.
- When autoscaling Service Fabric, the node types in your cluster are made of virtual machine scale sets at the back end, so you need to set up autoscale rules for each node type. Take into account the number of nodes that you must have before you set up autoscaling. The minimum number of nodes that you must have for the primary node type is driven by the reliability level you have chosen. For more information, see [scale a Service Fabric cluster in or out using autoscale rules](#).
- You can use the portal to link resources such as SQL Database instances and queues to a Cloud Service instance. This allows you to more easily access the separate manual and automatic scaling configuration options for each of the linked resources. For more information, see [How to: Link a resource to a cloud service](#).
- When you configure multiple policies and rules, they could conflict with each other. Autoscale uses the following conflict resolution rules to ensure that there is always a sufficient number of instances running:
 - Scale-out operations always take precedence over scale-in operations.
 - When scale-out operations conflict, the rule that initiates the largest increase in the number of instances takes precedence.
 - When scale in operations conflict, the rule that initiates the smallest decrease in the number of instances takes precedence.
- In an App Service Environment, any worker pool or front-end metrics can be used to define autoscale rules. For more information, see [Autoscaling and App Service Environment](#).

Application design considerations

Autoscaling isn't an instant solution. Simply adding resources to a system or running more instances of a process doesn't guarantee that the performance of the system will improve. Consider the following points when designing an autoscaling strategy:

- The system must be designed to be horizontally scalable. Avoid making assumptions about instance affinity; do not design solutions that require that the code is always running in a specific instance of a process. When scaling a cloud service or web site horizontally, don't assume that a series of requests from the same source will always be routed to the same instance. For the same reason, design services to be stateless to avoid requiring a series of requests from an application to always be routed to the same instance of a service. When designing a service that reads messages from a queue and processes them, don't make any assumptions about which instance of the service handles a specific message. Autoscaling could start additional instances of a service as the queue length grows. The [Competing Consumers pattern](#) describes how to handle this scenario.
- If the solution implements a long-running task, design this task to support both scaling out and scaling in. Without due care, such a task could prevent an instance of a process from being shut down cleanly when the system scales in, or it could lose data if the process is forcibly terminated. Ideally, refactor a long-running task and break up the processing that it performs into smaller, discrete chunks. The [Pipes and Filters pattern](#) provides an example of how you can achieve this.

- Alternatively, you can implement a checkpoint mechanism that records state information about the task at regular intervals, and save this state in durable storage that can be accessed by any instance of the process running the task. In this way, if the process is shut down, the work that it was performing can be resumed from the last checkpoint by using another instance.
- When background tasks run on separate compute instances, such as in worker roles of a cloud-services-hosted application, you may need to scale different parts of the application using different scaling policies. For example, you may need to deploy additional user interface (UI) compute instances without increasing the number of background compute instances, or the opposite of this. If you offer different levels of service (such as basic and premium service packages), you may need to scale out the compute resources for premium service packages more aggressively than those for basic service packages in order to meet SLAs.
- Consider using the length of the queue over which UI and background compute instances communicate as a criterion for your autoscaling strategy. This is the best indicator of an imbalance or difference between the current load and the processing capacity of the background task.
- If you base your autoscaling strategy on counters that measure business processes, such as the number of orders placed per hour or the average execution time of a complex transaction, ensure that you fully understand the relationship between the results from these types of counters and the actual compute capacity requirements. It may be necessary to scale more than one component or compute unit in response to changes in business process counters.
- To prevent a system from attempting to scale out excessively, and to avoid the costs associated with running many thousands of instances, consider limiting the maximum number of instances that can be automatically added. Most autoscaling mechanisms allow you to specify the minimum and maximum number of instances for a rule. In addition, consider gracefully degrading the functionality that the system provides if the maximum number of instances have been deployed, and the system is still overloaded.
- Keep in mind that autoscaling might not be the most appropriate mechanism to handle a sudden burst in workload. It takes time to provision and start new instances of a service or add resources to a system, and the peak demand may have passed by the time these additional resources have been made available. In this scenario, it may be better to throttle the service. For more information, see the [Throttling pattern](#).
- Conversely, if you do need the capacity to process all requests when the volume fluctuates rapidly, and cost isn't a major contributing factor, consider using an aggressive autoscaling strategy that starts additional instances more quickly. You can also use a scheduled policy that starts a sufficient number of instances to meet the maximum load before that load is expected.
- The autoscaling mechanism should monitor the autoscaling process, and log the details of each autoscaling event (what triggered it, what resources were added or removed, and when). If you create a custom autoscaling mechanism, ensure that it incorporates this capability. Analyze the information to help measure the effectiveness of the autoscaling strategy, and tune it if necessary. You can tune both in the short term, as the usage patterns become more obvious, and over the long term, as the business expands or the requirements of the application evolve. If an application reaches the upper limit defined for autoscaling, the mechanism might also alert an operator who could manually start additional resources if necessary. Note that under these circumstances the operator may also be responsible for manually removing these resources after the workload eases.

Related patterns and guidance

The following patterns and guidance may also be relevant to your scenario when implementing autoscaling:

- [Throttling pattern](#). This pattern describes how an application can continue to function and meet SLAs when an increase in demand places an extreme load on resources. Throttling can be used with autoscaling to prevent a system from being overwhelmed while the system scales out.

- [Competing Consumers pattern](#). This pattern describes how to implement a pool of service instances that can handle messages from any application instance. Autoscaling can be used to start and stop service instances to match the anticipated workload. This approach enables a system to process multiple messages concurrently to optimize throughput, improve scalability and availability, and balance the workload.
- [Monitoring and diagnostics](#). Instrumentation and telemetry are vital for gathering the information that can drive the autoscaling process.

Background jobs

3/10/2022 • 25 minutes to read • [Edit Online](#)

Many types of applications require background tasks that run independently of the user interface (UI). Examples include batch jobs, intensive processing tasks, and long-running processes such as workflows. Background jobs can be executed without requiring user interaction--the application can start the job and then continue to process interactive requests from users. This can help to minimize the load on the application UI, which can improve availability and reduce interactive response times.

For example, if an application is required to generate thumbnails of images that are uploaded by users, it can do this as a background job and save the thumbnail to storage when it is complete--without the user needing to wait for the process to be completed. In the same way, a user placing an order can initiate a background workflow that processes the order, while the UI allows the user to continue browsing the web app. When the background job is complete, it can update the stored orders data and send an email to the user that confirms the order.

When you consider whether to implement a task as a background job, the main criteria is whether the task can run without user interaction and without the UI needing to wait for the job to be completed. Tasks that require the user or the UI to wait while they are completed might not be appropriate as background jobs.

Types of background jobs

Background jobs typically include one or more of the following types of jobs:

- CPU-intensive jobs, such as mathematical calculations or structural model analysis.
- I/O-intensive jobs, such as executing a series of storage transactions or indexing files.
- Batch jobs, such as nightly data updates or scheduled processing.
- Long-running workflows, such as order fulfillment, or provisioning services and systems.
- Sensitive-data processing where the task is handed off to a more secure location for processing. For example, you might not want to process sensitive data within a web app. Instead, you might use a pattern such as the [Gatekeeper pattern](#) to transfer the data to an isolated background process that has access to protected storage.

Triggers

Background jobs can be initiated in several different ways. They fall into one of the following categories:

- **Event-driven triggers.** The task is started in response to an event, typically an action taken by a user or a step in a workflow.
- **Schedule-driven triggers.** The task is invoked on a schedule based on a timer. This might be a recurring schedule or a one-off invocation that is specified for a later time.

Event-driven triggers

Event-driven invocation uses a trigger to start the background task. Examples of using event-driven triggers include:

- The UI or another job places a message in a queue. The message contains data about an action that has taken place, such as the user placing an order. The background task listens on this queue and detects the arrival of a new message. It reads the message and uses the data in it as the input to the background job.
- The UI or another job saves or updates a value in storage. The background task monitors the storage and detects changes. It reads the data and uses it as the input to the background job.

- The UI or another job makes a request to an endpoint, such as an HTTPS URI, or an API that is exposed as a web service. It passes the data that is required to complete the background task as part of the request. The endpoint or web service invokes the background task, which uses the data as its input.

Typical examples of tasks that are suited to event-driven invocation include image processing, workflows, sending information to remote services, sending email messages, and provisioning new users in multitenant applications.

Schedule-driven triggers

Schedule-driven invocation uses a timer to start the background task. Examples of using schedule-driven triggers include:

- A timer that is running locally within the application or as part of the application's operating system invokes a background task on a regular basis.
- A timer that is running in a different application, such as Azure Logic Apps, sends a request to an API or web service on a regular basis. The API or web service invokes the background task.
- A separate process or application starts a timer that causes the background task to be invoked once after a specified time delay, or at a specific time.

Typical examples of tasks that are suited to schedule-driven invocation include batch-processing routines (such as updating related-products lists for users based on their recent behavior), routine data processing tasks (such as updating indexes or generating accumulated results), data analysis for daily reports, data retention cleanup, and data consistency checks.

If you use a schedule-driven task that must run as a single instance, be aware of the following:

- If the compute instance that is running the scheduler (such as a virtual machine using Windows scheduled tasks) is scaled, you will have multiple instances of the scheduler running. These could start multiple instances of the task.
- If tasks run for longer than the period between scheduler events, the scheduler may start another instance of the task while the previous one is still running.

Returning results

Background jobs execute asynchronously in a separate process, or even in a separate location, from the UI or the process that invoked the background task. Ideally, background tasks are "fire and forget" operations, and their execution progress has no impact on the UI or the calling process. This means that the calling process does not wait for completion of the tasks. Therefore, it cannot automatically detect when the task ends.

If you require a background task to communicate with the calling task to indicate progress or completion, you must implement a mechanism for this. Some examples are:

- Write a status indicator value to storage that is accessible to the UI or caller task, which can monitor or check this value when required. Other data that the background task must return to the caller can be placed into the same storage.
- Establish a reply queue that the UI or caller listens on. The background task can send messages to the queue that indicate status and completion. Data that the background task must return to the caller can be placed into the messages. If you are using Azure Service Bus, you can use the **ReplyTo** and **CorrelationId** properties to implement this capability.
- Expose an API or endpoint from the background task that the UI or caller can access to obtain status information. Data that the background task must return to the caller can be included in the response.
- Have the background task call back to the UI or caller through an API to indicate status at predefined points or on completion. This might be through events raised locally or through a publish-and-subscribe mechanism. Data that the background task must return to the caller can be included in the request or event

payload.

Hosting environment

You can host background tasks by using a range of different Azure platform services:

- [Azure Web Apps and WebJobs](#). You can use WebJobs to execute custom jobs based on a range of different types of scripts or executable programs within the context of a web app.
- [Azure Functions](#). You can use functions for background jobs that don't run for a long time. Another use case is if your workload is already hosted on App Service plan and is underutilized.
- [Azure Virtual Machines](#). If you have a Windows service or want to use the Windows Task Scheduler, it is common to host your background tasks within a dedicated virtual machine.
- [Azure Batch](#). Batch is a platform service that schedules compute-intensive work to run on a managed collection of virtual machines. It can automatically scale compute resources.
- [Azure Kubernetes Service \(AKS\)](#). Azure Kubernetes Service provides a managed hosting environment for Kubernetes on Azure.

The following sections describe each of these options in more detail, and include considerations to help you choose the appropriate option.

Azure Web Apps and WebJobs

You can use Azure WebJobs to execute custom jobs as background tasks within an Azure Web App. WebJobs run within the context of your web app as a continuous process. WebJobs also run in response to a trigger event from Azure Logic Apps or external factors, such as changes to storage blobs and message queues. Jobs can be started and stopped on demand, and shut down gracefully. If a continuously running WebJob fails, it is automatically restarted. Retry and error actions are configurable.

When you configure a WebJob:

- If you want the job to respond to an event-driven trigger, you should configure it as **Run continuously**. The script or program is stored in the folder named site/wwwroot/app_data/jobs/continuous.
- If you want the job to respond to a schedule-driven trigger, you should configure it as **Run on a schedule**. The script or program is stored in the folder named site/wwwroot/app_data/jobs/triggered.
- If you choose the **Run on demand** option when you configure a job, it will execute the same code as the **Run on a schedule** option when you start it.

Azure WebJobs run within the sandbox of the web app. This means that they can access environment variables and share information, such as connection strings, with the web app. The job has access to the unique identifier of the machine that is running the job. The connection string named **AzureWebJobsStorage** provides access to Azure storage queues, blobs, and tables for application data, and access to Service Bus for messaging and communication. The connection string named **AzureWebJobsDashboard** provides access to the job action log files.

Azure WebJobs have the following characteristics:

- **Security**: WebJobs are protected by the deployment credentials of the web app.
- **Supported file types**: You can define WebJobs by using command scripts (.cmd), batch files (.bat), PowerShell scripts (.ps1), bash shell scripts (.sh), PHP scripts (.php), Python scripts (.py), JavaScript code (.js), and executable programs (.exe, .jar, and more).
- **Deployment**: You can deploy scripts and executables by using the [Azure portal](#), by using [Visual Studio](#), by using the [Azure WebJobs SDK](#), or by copying them directly to the following locations:
 - For triggered execution: site/wwwroot/app_data/jobs/triggered/{job name}
 - For continuous execution: site/wwwroot/app_data/jobs/continuous/{job name}
- **Logging**: Console.Out is treated (marked) as INFO. Console.Error is treated as ERROR. You can access

monitoring and diagnostics information by using the Azure portal. You can download log files directly from the site. They are saved in the following locations:

- For triggered execution: Vfs/data/jobs/triggered/jobName
- For continuous execution: Vfs/data/jobs/continuous/jobName

- **Configuration:** You can configure WebJobs by using the portal, the REST API, and PowerShell. You can use a configuration file named settings.job in the same root directory as the job script to provide configuration information for a job. For example:

- { "stopping_wait_time": 60 }
- { "is_singleton": true }

Considerations

- By default, WebJobs scale with the web app. However, you can configure jobs to run on single instance by setting the **is_singleton** configuration property to **true**. Single instance WebJobs are useful for tasks that you do not want to scale or run as simultaneous multiple instances, such as reindexing, data analysis, and similar tasks.
- To minimize the impact of jobs on the performance of the web app, consider creating an empty Azure Web App instance in a new App Service plan to host long-running or resource-intensive WebJobs.

Azure Functions

An option that is similar to WebJobs is Azure Functions. This service is serverless that is most suitable for event-driven triggers that run for a short period. A function can also be used to run scheduled jobs through timer triggers, when configured to run at set times.

Azure Functions is not a recommended option for large, long-running tasks because they can cause unexpected timeout issues. However, depending on the hosting plan, they can be considered for schedule-driven triggers.

Considerations

If the background task is expected to run for a short duration in response to an event, consider running the task in a Consumption plan. The execution time is configurable up to a maximum time. A function that runs for longer costs more. Also CPU-intensive jobs that consume more memory can be more expensive. If you use additional triggers for services as part of your task, those are billed separately.

The Premium plan is more suitable if you have a high number of tasks that are short but expected to run continuously. This plan is more expensive because it needs more memory and CPU. The benefit is that you can use features such as virtual network integration.

The Dedicated plan is most suitable for background jobs if your workload already runs on it. If you have underutilized VMs, you can run it on the same VM and share compute costs.

For more information, see these articles:

- [Azure Functions hosting options](#)
- [Timer trigger for Azure Functions](#)

Azure Virtual Machines

Background tasks might be implemented in a way that prevents them from being deployed to Azure Web Apps, or these options might not be convenient. Typical examples are Windows services, and third-party utilities and executable programs. Another example might be programs written for an execution environment that is different than that hosting the application. For example, it might be a Unix or Linux program that you want to execute from a Windows or .NET application. You can choose from a range of operating systems for an Azure virtual machine, and run your service or executable on that virtual machine.

To help you choose when to use Virtual Machines, see [Azure App Services, Cloud Services and Virtual Machines comparison](#). For information about the options for Virtual Machines, see [Sizes for Windows virtual machines in Azure](#). For more information about the operating systems and prebuilt images that are available for Virtual

Machines, see [Azure Virtual Machines Marketplace](#).

To initiate the background task in a separate virtual machine, you have a range of options:

- You can execute the task on demand directly from your application by sending a request to an endpoint that the task exposes. This passes in any data that the task requires. This endpoint invokes the task.
- You can configure the task to run on a schedule by using a scheduler or timer that is available in your chosen operating system. For example, on Windows you can use Windows Task Scheduler to execute scripts and tasks. Or, if you have SQL Server installed on the virtual machine, you can use the SQL Server Agent to execute scripts and tasks.
- You can use Azure Logic Apps to initiate the task by adding a message to a queue that the task listens on, or by sending a request to an API that the task exposes.

See the earlier section [Triggers](#) for more information about how you can initiate background tasks.

Considerations

Consider the following points when you are deciding whether to deploy background tasks in an Azure virtual machine:

- Hosting background tasks in a separate Azure virtual machine provides flexibility and allows precise control over initiation, execution, scheduling, and resource allocation. However, it will increase runtime cost if a virtual machine must be deployed just to run background tasks.
- There is no facility to monitor the tasks in the Azure portal and no automated restart capability for failed tasks--although you can monitor the basic status of the virtual machine and manage it by using the [Azure Resource Manager Cmdlets](#). However, there are no facilities to control processes and threads in compute nodes. Typically, using a virtual machine will require additional effort to implement a mechanism that collects data from instrumentation in the task, and from the operating system in the virtual machine. One solution that might be appropriate is to use the [System Center Management Pack for Azure](#).
- You might consider creating monitoring probes that are exposed through HTTP endpoints. The code for these probes could perform health checks, collect operational information and statistics--or collate error information and return it to a management application. For more information, see the [Health Endpoint Monitoring pattern](#).

For more information, see:

- [Virtual Machines](#)
- [Azure Virtual Machines FAQ](#)

Azure Batch

Consider [Azure Batch](#) if you need to run large, parallel high-performance computing (HPC) workloads across tens, hundreds, or thousands of VMs.

The Batch service provisions the VMs, assign tasks to the VMs, runs the tasks, and monitors the progress. Batch can automatically scale out the VMs in response to the workload. Batch also provides job scheduling. Azure Batch supports both Linux and Windows VMs.

Considerations

Batch works well with intrinsically parallel workloads. It can also perform parallel calculations with a reduce step at the end, or run [Message Passing Interface \(MPI\) applications](#) for parallel tasks that require message passing between nodes.

An Azure Batch job runs on a pool of nodes (VMs). One approach is to allocate a pool only when needed and then delete it after the job completes. This maximizes utilization, because nodes are not idle, but the job must wait for nodes to be allocated. Alternatively, you can create a pool ahead of time. That approach minimizes the time that it takes for a job to start, but can result in having nodes that sit idle. For more information, see [Pool and compute node lifetime](#).

For more information, see:

- [What is Azure Batch?](#)
- [Develop large-scale parallel compute solutions with Batch](#)
- [Batch and HPC solutions for large-scale computing workloads](#)

Azure Kubernetes Service

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, which makes it easy to deploy and manage containerized applications.

Containers can be useful for running background jobs. Some of the benefits include:

- Containers support high-density hosting. You can isolate a background task in a container, while placing multiple containers in each VM.
- The container orchestrator handles internal load balancing, configuring the internal network, and other configuration tasks.
- Containers can be started and stopped as needed.
- Azure Container Registry allows you to register your containers inside Azure boundaries. This comes with security, privacy, and proximity benefits.

Considerations

- Requires an understanding of how to use a container orchestrator. Depending on the skill set of your DevOps team, this may or may not be an issue.

For more information, see:

- [Overview of containers in Azure](#)
- [Introduction to private Docker container registries](#)

Partitioning

If you decide to include background tasks within an existing compute instance, you must consider how this will affect the quality attributes of the compute instance and the background task itself. These factors will help you to decide whether to colocate the tasks with the existing compute instance or separate them out into a separate compute instance:

- **Availability:** Background tasks might not need to have the same level of availability as other parts of the application, in particular the UI and other parts that are directly involved in user interaction. Background tasks might be more tolerant of latency, retried connection failures, and other factors that affect availability because the operations can be queued. However, there must be sufficient capacity to prevent the backup of requests that could block queues and affect the application as a whole.
- **Scalability:** Background tasks are likely to have a different scalability requirement than the UI and the interactive parts of the application. Scaling the UI might be necessary to meet peaks in demand, while outstanding background tasks might be completed during less busy times by fewer compute instances.
- **Resiliency:** Failure of a compute instance that just hosts background tasks might not fatally affect the application as a whole if the requests for these tasks can be queued or postponed until the task is available again. If the compute instance and/or tasks can be restarted within an appropriate interval, users of the application might not be affected.
- **Security:** Background tasks might have different security requirements or restrictions than the UI or other parts of the application. By using a separate compute instance, you can specify a different security environment for the tasks. You can also use patterns such as Gatekeeper to isolate the background compute instances from the UI in order to maximize security and separation.

- **Performance:** You can choose the type of compute instance for background tasks to specifically match the performance requirements of the tasks. This might mean using a less expensive compute option if the tasks do not require the same processing capabilities as the UI, or a larger instance if they require additional capacity and resources.
- **Manageability:** Background tasks might have a different development and deployment rhythm from the main application code or the UI. Deploying them to a separate compute instance can simplify updates and versioning.
- **Cost:** Adding compute instances to execute background tasks increases hosting costs. You should carefully consider the trade-off between additional capacity and these extra costs.

For more information, see the [Leader Election pattern](#) and the [Competing Consumers pattern](#).

Conflicts

If you have multiple instances of a background job, it is possible that they will compete for access to resources and services, such as databases and storage. This concurrent access can result in resource contention, which might cause conflicts in availability of the services and in the integrity of data in storage. You can resolve resource contention by using a pessimistic locking approach. This prevents competing instances of a task from concurrently accessing a service or corrupting data.

Another approach to resolve conflicts is to define background tasks as a singleton, so that there is only ever one instance running. However, this eliminates the reliability and performance benefits that a multiple-instance configuration can provide. This is especially true if the UI can supply sufficient work to keep more than one background task busy.

It is vital to ensure that the background task can automatically restart and that it has sufficient capacity to cope with peaks in demand. You can achieve this by allocating a compute instance with sufficient resources, by implementing a queueing mechanism that can store requests for later execution when demand decreases, or by using a combination of these techniques.

Coordination

The background tasks might be complex and might require multiple individual tasks to execute to produce a result or to fulfill all the requirements. It is common in these scenarios to divide the task into smaller discreet steps or subtasks that can be executed by multiple consumers. Multistep jobs can be more efficient and more flexible because individual steps might be reusable in multiple jobs. It is also easy to add, remove, or modify the order of the steps.

Coordinating multiple tasks and steps can be challenging, but there are three common patterns that you can use to guide your implementation of a solution:

- **Decomposing a task into multiple reusable steps.** An application might be required to perform a variety of tasks of varying complexity on the information that it processes. A straightforward but inflexible approach to implementing this application might be to perform this processing as a monolithic module. However, this approach is likely to reduce the opportunities for refactoring the code, optimizing it, or reusing it if parts of the same processing are required elsewhere within the application. For more information, see the [Pipes and Filters pattern](#).
- **Managing execution of the steps for a task.** An application might perform tasks that comprise a number of steps (some of which might invoke remote services or access remote resources). The individual steps might be independent of each other, but they are orchestrated by the application logic that implements the task. For more information, see [Scheduler Agent Supervisor pattern](#).
- **Managing recovery for task steps that fail.** An application might need to undo the work that is

performed by a series of steps (which together define an eventually consistent operation) if one or more of the steps fail. For more information, see the [Compensating Transaction pattern](#).

Resiliency considerations

Background tasks must be resilient in order to provide reliable services to the application. When you are planning and designing background tasks, consider the following points:

- Background tasks must be able to gracefully handle restarts without corrupting data or introducing inconsistency into the application. For long-running or multistep tasks, consider using *check pointing* by saving the state of jobs in persistent storage, or as messages in a queue if this is appropriate. For example, you can persist state information in a message in a queue and incrementally update this state information with the task progress so that the task can be processed from the last known good checkpoint--instead of restarting from the beginning. When using Azure Service Bus queues, you can use message sessions to enable the same scenario. Sessions allow you to save and retrieve the application processing state by using the [SetState](#) and [GetState](#) methods. For more information about designing reliable multistep processes and workflows, see the [Scheduler Agent Supervisor pattern](#).
- When you use queues to communicate with background tasks, the queues can act as a buffer to store requests that are sent to the tasks while the application is under higher than usual load. This allows the tasks to catch up with the UI during less busy periods. It also means that restarts will not block the UI. For more information, see the [Queue-Based Load Leveling pattern](#). If some tasks are more important than others, consider implementing the [Priority Queue pattern](#) to ensure that these tasks run before less important ones.
- Background tasks that are initiated by messages or process messages must be designed to handle inconsistencies, such as messages arriving out of order, messages that repeatedly cause an error (often referred to as *poison messages*), and messages that are delivered more than once. Consider the following:
 - Messages that must be processed in a specific order, such as those that change data based on the existing data value (for example, adding a value to an existing value), might not arrive in the original order in which they were sent. Alternatively, they might be handled by different instances of a background task in a different order due to varying loads on each instance. Messages that must be processed in a specific order should include a sequence number, key, or some other indicator that background tasks can use to ensure that they are processed in the correct order. If you are using Azure Service Bus, you can use message sessions to guarantee the order of delivery. However, it is usually more efficient, where possible, to design the process so that the message order is not important.
 - Typically, a background task will peek at messages in the queue, which temporarily hides them from other message consumers. Then it deletes the messages after they have been successfully processed. If a background task fails when processing a message, that message will reappear on the queue after the peek time-out expires. It will be processed by another instance of the task or during the next processing cycle of this instance. If the message consistently causes an error in the consumer, it will block the task, the queue, and eventually the application itself when the queue becomes full. Therefore, it is vital to detect and remove poison messages from the queue. If you are using Azure Service Bus, messages that cause an error can be moved automatically or manually to an associated dead letter queue.
 - Queues are guaranteed at *least once* delivery mechanisms, but they might deliver the same message more than once. In addition, if a background task fails after processing a message but before deleting it from the queue, the message will become available for processing again. Background tasks should be idempotent, which means that processing the same message more than once does not cause an error or inconsistency in the application's data. Some operations are

naturally idempotent, such as setting a stored value to a specific new value. However, operations such as adding a value to an existing stored value without checking that the stored value is still the same as when the message was originally sent will cause inconsistencies. Azure Service Bus queues can be configured to automatically remove duplicated messages.

- Some messaging systems, such as Azure storage queues and Azure Service Bus queues, support a de-queue count property that indicates the number of times a message has been read from the queue. This can be useful in handling repeated and poison messages. For more information, see [Asynchronous Messaging Primer](#) and [Idempotency Patterns](#).

Scaling and performance considerations

Background tasks must offer sufficient performance to ensure they do not block the application, or cause inconsistencies due to delayed operation when the system is under load. Typically, performance is improved by scaling the compute instances that host the background tasks. When you are planning and designing background tasks, consider the following points around scalability and performance:

- Azure supports autoscaling (both scaling out and scaling back in) based on current demand and load or on a predefined schedule, for Web Apps and Virtual Machines hosted deployments. Use this feature to ensure that the application as a whole has sufficient performance capabilities while minimizing runtime costs.
- Where background tasks have a different performance capability from the other parts of an application (for example, the UI or components such as the data access layer), hosting the background tasks together in a separate compute service allows the UI and background tasks to scale independently to manage the load. If multiple background tasks have significantly different performance capabilities from each other, consider dividing them and scaling each type independently. However, note that this might increase runtime costs.
- Simply scaling the compute resources might not be sufficient to prevent loss of performance under load. You might also need to scale storage queues and other resources to prevent a single point of the overall processing chain from becoming a bottleneck. Also, consider other limitations, such as the maximum throughput of storage and other services that the application and the background tasks rely on.
- Background tasks must be designed for scaling. For example, they must be able to dynamically detect the number of storage queues in use in order to listen on or send messages to the appropriate queue.
- By default, WebJobs scale with their associated Azure Web Apps instance. However, if you want a WebJob to run as only a single instance, you can create a `Settings.job` file that contains the JSON data `{ "is_singleton": true }`. This forces Azure to only run one instance of the WebJob, even if there are multiple instances of the associated web app. This can be a useful technique for scheduled jobs that must run as only a single instance.

Related patterns

- [Compute Partitioning Guidance](#)

Message encoding considerations

3/10/2022 • 11 minutes to read • [Edit Online](#)

Many cloud applications use asynchronous messages to exchange information between components of the system. An important aspect of messaging is the format used to encode the payload data. After you [choose a messaging technology](#), the next step is to define how the messages will be encoded. There are many options available, but the right choice depends on your use case.

This article describes some of the considerations.

Message exchange needs

A message exchange between a producer and a consumer needs:

- A shape or structure that defines the payload of the message.
- An encoding format to represent the payload.
- Serialization libraries to read and write the encoded payload.

The producer of the message defines the message shape based on the business logic and the information it wants to send to the consumer(s). To structure the shape, divide the information into discrete or related subjects (fields). Decide the characteristics of the values for those fields. Consider: What is the most efficient datatype? Will the payload always have certain fields? Will the payload have a single record or a repeated set of values?

Then, choose an encoding format depending on your need. Certain factors include the ability to create highly structured data if you need it, time taken to encode and transfer the message, and the ability to parse the payload. Depending on the encoding format, choose a serialization library that is well supported.

A consumer of the message must be aware of those decisions so that it knows how to read incoming messages.

To transfer messages, the producer serializes the message to an encoding format. At the receiving end, the consumer deserializes the payload to use the data. This way both entities share the model and as long as the shape doesn't change, messaging continues without issues. When the contract changes, the encoding format should be capable of handling the change without breaking the consumer.

Some encoding formats such as JSON are self-describing, meaning they can be parsed without referencing a schema. However, such formats tend to yield larger messages. With other formats, the data may not be parsed as easily but the messages are compact. This article highlights some factors that can help you choose a format.

Encoding format considerations

The encoding format defines how a set of structured data is represented as bytes. The type of message can influence the format choice. Messages related to business transactions most likely will contain highly structured data. Also, you may want to retrieve it later for auditing purposes. For a stream of events, you might want to read a sequence of records as quickly as possible and store it for statistical analysis.

Here are some points to consider when choosing an encoding format.

Human readability

Message encoding can be broadly divided into text-based and binary formats.

With text-based encoding, the message payload is in plain text and therefore can be inspected by a person without using any code libraries. Human readable formats are suitable for archival data. Also, because a human can read the payload, text-based formats are easier to debug and send to logs for troubleshooting errors.

The downside is that the payload tends to be larger. A common text-based format is JSON.

Encoding size

Message size impacts network I/O performance across the wire. Binary formats are more compact than text-based formats. Binary formats require serialization/deserialization libraries. The payload can't be read unless it's decoded.

Use a binary format if you want to reduce wire footprint and transfer messages faster. This category of format is recommended in scenarios where storage or network bandwidth is a concern. Options for binary formats include Apache Avro, Google Protocol Buffers (protobuf), MessagePack, and Concise Binary Object Representation (CBOR). The pros and cons of those formats are described in [this section](#).

The disadvantage is that the payload isn't human readable. Most binary formats use complex systems that can be costly to maintain. Also, they need specialized libraries to decode, which may not be supported if you want to retrieve archival data.

Understanding the payload

A message payload arrives as a sequence of bytes. To parse this sequence, the consumer must have access to metadata that describes the data fields in the payload. There are two main approaches for storing and distributing metadata:

Tagged metadata. In some encodings, notably JSON, fields are tagged with the data type and identifier, within the body of the message. These formats are *self-describing* because they can be parsed into a dictionary of values without referring to a schema. One way for the consumer to understand the fields is to query for expected values. For example, the producer sends a payload in JSON. The consumer parses the JSON into a dictionary and checks the existence of fields to understand the payload. Another way is for the consumer to apply a data model shared by the producer. For example, if you are using a statically typed language, many JSON serialization libraries can parse a JSON string into a typed class.

Schema. A schema formally defines the structure and data fields of a message. In this model, the producer and consumer have a contract through a well-defined schema. The schema can define the data types, required/optional fields, version information, and the structure of the payload. The producer sends the payload as per the writer schema. The consumer receives the payload by applying a reader schema. The message is serialized/deserialized by using the encoding-specific libraries. There are two ways to distribute schemas:

- Store the schema as a preamble or header in the message but separate from the payload.
- Store the schema externally.

Some encoding formats define the schema and use tools that generate classes from the schema. The producer and consumer use those classes and libraries to serialize and deserialize the payload. The libraries also provide compatibility checks between the writer and reader schema. Both protobuf and Apache Avro follow that approach. The key difference is that protobuf has a language-agnostic schema definition but Avro uses compact JSON. Another difference is in the way both formats provide compatibility checks between reader and writer schemas.

Another way to store the schema externally in a schema registry. The message contains a reference to the schema and the payload. The producer sends the schema identifier in the message and the consumer retrieves the schema by specifying that identifier from an external store. Both parties use format-specific library to read and write messages. Apart from storing the schema a registry can provide compatibility checks to make sure the contract between the producer and consumer isn't broken as the schema evolves.

Before choosing an approach, decide what is more important: the transfer data size or the ability to parse the archived data later.

Storing the schema along with the payload yields a larger encoding size and is preferred for intermittent messages. Choose this approach if transferring smaller chunks of bytes is crucial or you expect a sequence of

records. The cost of maintaining an external schema store can be high.

However, if on-demand decoding of the payload is more important than size, including the schema with the payload or the tagged metadata approach guarantees decoding afterwards. There might be a significant increase in message size and may impact the cost of storage.

Schema versioning

As business requirements change, the shape is expected to change, and the schema will evolve. Versioning allows the producer to indicate schema updates that might include new features. There are two aspects to versioning:

- The consumer should be aware of the changes.

One way is for the consumer to check all fields to determine whether the schema has changed. Another way is for the producer to publish a schema version number with the message. When the schema evolves, the producer increments the version.

- Changes must not affect or break the business logic of consumers.

Suppose a field is added to an existing schema. If consumers using the new version get a payload as per the old version, their logic might break if they are not able to overlook the lack of the new field.

Considering the reverse case, suppose a field is removed in the new schema. Consumers using the old schema might not be able to read the data.

Encoding formats such as Avro offer the ability to define default values. In the preceding example, if the field is added with a default value, the missing field will be populated with the default value. Other formats such as protobuf provide similar functionality through required and optional fields.

Payload structure

Consider the way data is arranged in the payload. Is it a sequence of records or a discrete single payload? The payload structure can be categorized into one of these models:

- Array/dictionary/value: Defines entries that hold values in one or multi-dimensional arrays. Entries have unique key-value pairs. It can be extended to represent the complex structures. Some examples include, JSON, Apache Avro, and MessagePack.

This layout is suitable if messages are individual encoded with different schemas. If you have multiple records, the payload can get overly redundant causing the payload to bloat.

- Tabular data: Information is divided into rows and columns. Each column indicates a field, or the subject of the information and each row contains values for those fields. This layout is efficient for a repeating set of information, such as time series data.

CSV is one of the simplest text-based formats. It presents data as a sequence of records with a common header. For binary encoding, Apache Avro has a preamble is similar to a CSV header but generate compact encoding size.

Library support

Consider using well-known formats over a proprietary model.

Well-known formats are supported through libraries that are universally supported by the community. With specialized formats, you need specific libraries. Your business logic might have to work around some of the API design choices provided by the libraries.

For schema-based format, choose an encoding library that makes compatibility checks between the reader and writer schema. Certain encoding libraries, such as Apache Avro, expect the consumer to specify both writer and the reader schema before deserializing the message. This check ensures that the consumer is aware of the schema versions.

Interoperability

Your choice of formats might depend on the particular workload or technology ecosystem.

For example:

- Azure Stream Analytics has native support for JSON, CSV, and Avro. When using Stream Analytics, it makes sense to choose one of these formats if possible. If not, you can provide a [custom deserializer](#), but this adds some additional complexity to your solution.
- JSON is a standard interchange format for HTTP REST APIs. If your application receives JSON payloads from clients and then places these onto a message queue for asynchronous processing, it might make sense to use JSON for the messaging, rather than re-encode into a different format.

These are just two examples of interoperability considerations. In general, standardized formats will be more interoperable than custom formats. In text-based options, JSON is one of the most interoperable.

Choices for encoding formats

Here are some popular encoding formats. Factor in the considerations before you choose a format.

JSON

[JSON](#) is an open standard (IETF [RFC8259](#)). It's a text-based format that follows the array/dictionary/value model.

JSON can be used for tagging metadata and you can parse the payload without a schema. JSON supports the option to specify optional fields, which helps with forward and backward compatibility.

The biggest advantage is that its universally available. It's most interoperable and the default encoding format for many messaging services.

Being a text-based format, it isn't efficient over the wire and not an ideal choice in cases where storage is a concern. If you're returning cached items directly to a client via HTTP, storing JSON could save the cost of deserializing from another format and then serializing to JSON.

Use JSON for single-record messages or for a sequence of messages in which each message has a different schema. Avoid using JSON for a sequence of records, such as for time-series data.

There are other variations of JSON such as [BSON](#), which is a binary encoding aligned to work with MongoDB.

Comma-Separated Values (CSV)

CSV is a text-based tabular format. The header of the table indicates the fields. It's a preferred choice where the message contains a set of records.

The disadvantage is lack of standardization. There are many ways of expressing separators, headers, and empty fields.

Protocol Buffers (protobuf)

[Google Protocol Buffers](#) (or protobuf) is a serialization format that uses strongly typed definition files to define schemas in key/value pairs. These definition files are then compiled to language-specific classes that are used for serializing and deserializing messages.

The message contains a compressed binary small payload, which results in faster transfer. The downside is the payload isn't human readable. Also, because the schema is external, it's not recommended for cases where you have to retrieve archived data.

Apache Avro

[Apache Avro](#) is a binary serialization format that uses definition file similar to protobuf but there isn't a compilation step. Instead, serialized data always includes a schema preamble.

The preamble can hold the header or a schema identifier. Because of the smaller encoding size, Avro is recommended for streaming data. Also, because it has a header that applies to a set of records, it's a good choice for tabular data.

MessagePack

[MessagePack](#) is a binary serialization format that is designed to be compact for transmission over the wire. There are no message schemas or message type checking. This format isn't recommended for bulk storage.

CBOR

[Concise Binary Object Representation \(CBOR\) \(Specification\)](#) is a binary format that offers small encoding size. The advantage of CBOR over MessagePack is that its compliant with IETF in RFC7049.

Next steps

- Understand [messaging design patterns](#) for cloud applications.

Retry guidance for Azure services

3/10/2022 • 41 minutes to read • [Edit Online](#)

Most Azure services and client SDKs include a retry mechanism. However, these differ because each service has different characteristics and requirements, and so each retry mechanism is tuned to a specific service. This guide summarizes the retry mechanism features for the majority of Azure services, and includes information to help you use, adapt, or extend the retry mechanism for that service.

For general guidance on handling transient faults, and retrying connections and operations against services and resources, see [Retry guidance](#).

The following table summarizes the retry features for the Azure services described in this guidance.

Service	Retry Capabilities	Policy Configuration	Scope	Telemetry Features
Azure Active Directory	Native in MSAL library	Embedded into MSAL library	Internal	None
Cosmos DB	Native in service	Non-configurable	Global	TraceSource
Data Lake Store	Native in client	Non-configurable	Individual operations	None
Event Hubs	Native in client	Programmatic	Client	None
IoT Hub	Native in client SDK	Programmatic	Client	None
Azure Cache for Redis	Native in client	Programmatic	Client	TextWriter
Search	Native in client	Programmatic	Client	ETW or Custom
Service Bus	Native in client	Programmatic	Namespace Manager, Messaging Factory, and Client	ETW
Service Fabric	Native in client	Programmatic	Client	None
SQL Database with ADO.NET	Polly	Declarative and programmatic	Single statements or blocks of code	Custom
SQL Database with Entity Framework	Native in client	Programmatic	Global per AppDomain	None
SQL Database with Entity Framework Core	Native in client	Programmatic	Global per AppDomain	None
Storage	Native in client	Programmatic	Client and individual operations	TraceSource

NOTE

For most of the Azure built-in retry mechanisms, there is currently no way apply a different retry policy for different types of error or exception. You should configure a policy that provides the optimum average performance and availability. One way to fine-tune the policy is to analyze log files to determine the type of transient faults that are occurring.

Azure Active Directory

Azure Active Directory (Azure AD) is a comprehensive identity and access management cloud solution that combines core directory services, advanced identity governance, security, and application access management. Azure AD also offers developers an identity management platform to deliver access control to their applications, based on centralized policy and rules.

NOTE

For retry guidance on Managed Service Identity endpoints, see [How to use an Azure VM Managed Service Identity \(MSI\) for token acquisition](#).

Retry mechanism

There is a built-in retry mechanism for Azure Active Directory in the [Microsoft Authentication Library \(MSAL\)](#). To avoid unexpected lockouts, we recommend that third-party libraries and application code do **not** retry failed connections, but allow MSAL to handle retries.

Retry usage guidance

Consider the following guidelines when using Azure Active Directory:

- When possible, use the MSAL library and the built-in support for retries.
- If you are using the REST API for Azure Active Directory, retry the operation if the result code is 429 (Too Many Requests) or an error in the 5xx range. Do not retry for any other errors.
- For 429 errors, only retry after the time indicated in the **Retry-After** header.
- For 5xx errors, use exponential back-off, with the first retry at least 5 seconds after the response.
- Do not retry on errors other than 429 and 5xx.

More information

- [Microsoft Authentication Library \(MSAL\)](#)

Cosmos DB

Cosmos DB is a fully managed multi-model database that supports schemaless JSON data. It offers configurable and reliable performance, native JavaScript transactional processing, and is built for the cloud with elastic scale.

Retry mechanism

The `CosmosClient` class automatically retries failed attempts. To set the number of retries and the maximum wait time, configure `CosmosClientOptions`. Exceptions that the client raises are either beyond the retry policy or are not transient errors. If Cosmos DB throttles the client, it returns an HTTP 429 error. Check the status code in the `CosmosException` class.

Policy configuration

The following table shows the default settings for the `CosmosClientOptions` class.

Setting	Default Value	Description
MaxRetryAttemptsOnRateLimitedRequests	9	The maximum number of retries if the request fails because Cosmos DB applied rate limiting on the client.
MaxRetryWaitTimeOnRateLimitedRequests	30	The maximum retry time in seconds for the Azure Cosmos DB service.

Example

```
CosmosClient cosmosClient = new CosmosClient("connection-string", new CosmosClientOptions()
{
    MaxRetryAttemptsOnRateLimitedRequests = 5,
    MaxRetryWaitTimeOnRateLimitedRequests = TimeSpan.FromSeconds(15)
});
```

Telemetry

Retry attempts are logged as unstructured trace messages through a .NET **TraceSource**. You must configure a **TraceListener** to capture the events and write them to a suitable destination log.

For example, if you add the following to your App.config file, traces will be generated in a text file in the same location as the executable:

```
<configuration>
  <system.diagnostics>
    <switches>
      <add name="SourceSwitch" value="Verbose"/>
    </switches>
    <sources>
      <source name="DocDBTrace" switchName="SourceSwitch" switchType="System.Diagnostics.SourceSwitch" >
        <listeners>
          <add name="MyTextListener" type="System.Diagnostics.TextWriterTraceListener"
traceOutputOptions="DateTime,ProcessId,ThreadId" initializeData="CosmosDBTrace.txt"></add>
        </listeners>
      </source>
    </sources>
  </system.diagnostics>
</configuration>
```

Event Hubs

Azure Event Hubs is a hyperscale telemetry ingestion service that collects, transforms, and stores millions of events.

Retry mechanism

Retry behavior in the Azure Event Hubs Client Library is controlled by the `RetryPolicy` property on the `EventHubClient` class. The default policy retries with exponential backoff when Azure Event Hub returns a transient `EventHubsException` or an `OperationCanceledException`. Default retry policy for Event Hubs is to retry up to 9 times with an exponential back-off time of up to 30 seconds .

Example

```
EventHubClient client = EventHubClient.CreateFromConnectionString("[event_hub_connection_string]");
client.RetryPolicy = RetryPolicy.Default;
```

More information

[.NET Standard client library for Azure Event Hubs](#)

IoT Hub

Azure IoT Hub is a service for connecting, monitoring, and managing devices to develop Internet of Things (IoT) applications.

Retry mechanism

The Azure IoT device SDK can detect errors in the network, protocol, or application. Based on the error type, the SDK checks whether a retry needs to be performed. If the error is *recoverable*, the SDK begins to retry using the configured retry policy.

The default retry policy is *exponential back-off with random jitter*, but it can be configured.

Policy configuration

Policy configuration differs by language. For more details, see [IoT Hub retry policy configuration](#).

More information

- [IoT Hub retry policy](#)
- [Troubleshoot IoT Hub device disconnection](#)

Azure Cache for Redis

Azure Cache for Redis is a fast data access and low latency cache service based on the popular open-source Redis cache. It is secure, managed by Microsoft, and is accessible from any application in Azure.

The guidance in this section is based on using the StackExchange.Redis client to access the cache. A list of other suitable clients can be found on the [Redis website](#), and these may have different retry mechanisms.

Note that the StackExchange.Redis client uses multiplexing through a single connection. The recommended usage is to create an instance of the client at application startup and use this instance for all operations against the cache. For this reason, the connection to the cache is made only once, and so all of the guidance in this section is related to the retry policy for this initial connection—and not for each operation that accesses the cache.

Retry mechanism

The StackExchange.Redis client uses a connection manager class that is configured through a set of options, including:

- **ConnectRetry**. The number of times a failed connection to the cache will be retried.
- **ReconnectRetryPolicy**. The retry strategy to use.
- **ConnectTimeout**. The maximum waiting time in milliseconds.

Policy configuration

Retry policies are configured programmatically by setting the options for the client before connecting to the cache. This can be done by creating an instance of the **ConfigurationOptions** class, populating its properties, and passing it to the **Connect** method.

The built-in classes support linear (constant) delay and exponential backoff with randomized retry intervals. You can also create a custom retry policy by implementing the **IReconnectRetryPolicy** interface.

The following example configures a retry strategy using exponential backoff.

```

var deltaBackOffInMilliseconds = TimeSpan.FromSeconds(5).TotalMilliseconds;
var maxDeltaBackOffInMilliseconds = TimeSpan.FromSeconds(20).TotalMilliseconds;
var options = new ConfigurationOptions
{
    EndPoints = {"localhost"},
    ConnectRetry = 3,
    ReconnectRetryPolicy = new ExponentialRetry(deltaBackOffInMilliseconds, maxDeltaBackOffInMilliseconds),
    ConnectTimeout = 2000
};
ConnectionMultiplexer redis = ConnectionMultiplexer.Connect(options, writer);

```

Alternatively, you can specify the options as a string, and pass this to the **Connect** method. The **ReconnectRetryPolicy** property cannot be set this way, only through code.

```

var options = "localhost,connectRetry=3,connectTimeout=2000";
ConnectionMultiplexer redis = ConnectionMultiplexer.Connect(options, writer);

```

You can also specify options directly when you connect to the cache.

```

var conn = ConnectionMultiplexer.Connect("redis0:6380,redis1:6380,connectRetry=3");

```

For more information, see [Stack Exchange Redis Configuration](#) in the StackExchange.Redis documentation.

The following table shows the default settings for the built-in retry policy.

CONTEXT	SETTING	DEFAULT VALUE (V 1.2.2)	MEANING
ConfigurationOptions	ConnectRetry ConnectTimeout SyncTimeout ReconnectRetryPolicy	3 Maximum 5000 ms plus SyncTimeout 1000 LinearRetry 5000 ms	The number of times to repeat connect attempts during the initial connection operation. Timeout (ms) for connect operations. Not a delay between retry attempts. Time (ms) to allow for synchronous operations. Retry every 5000 ms.

NOTE

For synchronous operations, `SyncTimeout` can add to the end-to-end latency, but setting the value too low can cause excessive timeouts. See [How to troubleshoot Azure Cache for Redis](#). In general, avoid using synchronous operations, and use asynchronous operations instead. For more information, see [Pipelines and Multiplexers](#).

Retry usage guidance

Consider the following guidelines when using Azure Cache for Redis:

- The StackExchange Redis client manages its own retries, but only when establishing a connection to the cache when the application first starts. You can configure the connection timeout, the number of retry attempts, and the time between retries to establish this connection, but the retry policy does not apply to operations against the cache.
- Instead of using a large number of retry attempts, consider falling back by accessing the original data source instead.

Telemetry

You can collect information about connections (but not other operations) using a [TextWriter](#).

```
var writer = new StringWriter();
ConnectionMultiplexer redis = ConnectionMultiplexer.Connect(options, writer);
```

An example of the output this generates is shown below.

```
localhost:6379,connectTimeout=2000,connectRetry=3
1 unique nodes specified
Requesting tie-break from localhost:6379 > __Booksleeve_TieBreak...
Allowing endpoints 00:00:02 to respond...
localhost:6379 faulted: SocketFailure on PING
localhost:6379 failed to nominate (Faulted)
> UnableToResolvePhysicalConnection on GET
No masters detected
localhost:6379: Standalone v2.0.0, master; keep-alive: 00:01:00; int: Connecting; sub: Connecting; not in
use: DidNotRespond
localhost:6379: int ops=0, qu=0, qs=0, qc=1, wr=0, sync=1, socks=2; sub ops=0, qu=0, qs=0, qc=0, wr=0,
socks=2
Circular op-count snapshot; int: 0 (0.00 ops/s; spans 10s); sub: 0 (0.00 ops/s; spans 10s)
Sync timeouts: 0; fire and forget: 0; last heartbeat: -1s ago
resetting failing connections to retry...
retrying; attempts left: 2...
...
```

Examples

The following code example configures a constant (linear) delay between retries when initializing the `StackExchange.Redis` client. This example shows how to set the configuration using a `ConfigurationOptions` instance.

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using StackExchange.Redis;

namespace RetryCodeSamples
{
    class CacheRedisCodeSamples
    {
        public async static Task Samples()
        {
            var writer = new StringWriter();
            {
                try
                {
                    var retryTimeInMilliseconds = TimeSpan.FromSeconds(4).TotalMilliseconds; // delay
between retries

                    // Using object-based configuration.
                    var options = new ConfigurationOptions
                    {
                        EndPoints = { "localhost" },
                        ConnectRetry = 3,
                        ReconnectRetryPolicy = new LinearRetry(retryTimeInMilliseconds)
                    };
                    ConnectionMultiplexer redis = ConnectionMultiplexer.Connect(options, writer);

                    // Store a reference to the multiplexer for use in the application.
                }
                catch
                {
                    Console.WriteLine(writer.ToString());
                    throw;
                }
            }
        }
    }
}

```

The next example sets the configuration by specifying the options as a string. The connection timeout is the maximum period of time to wait for a connection to the cache, not the delay between retry attempts. Note that the **ReconnectRetryPolicy** property can only be set by code.

```

using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using StackExchange.Redis;

namespace RetryCodeSamples
{
    class CacheRedisCodeSamples
    {
        public async static Task Samples()
        {
            var writer = new StringWriter();
            {
                try
                {
                    // Using string-based configuration.
                    var options = "localhost,connectRetry=3,connectTimeout=2000";
                    ConnectionMultiplexer redis = ConnectionMultiplexer.Connect(options, writer);

                    // Store a reference to the multiplexer for use in the application.
                }
                catch
                {
                    Console.WriteLine(writer.ToString());
                    throw;
                }
            }
        }
    }
}

```

For more examples, see [Configuration](#) on the project website.

More information

- [Redis website](#)

Azure Search

Azure Search can be used to add powerful and sophisticated search capabilities to a website or application, quickly and easily tune search results, and construct rich and fine-tuned ranking models.

Retry mechanism

Retry behavior in the Azure Search SDK is controlled by the `SetRetryPolicy` method on the [SearchServiceClient](#) and [SearchIndexClient](#) classes. The default policy retries with exponential backoff when Azure Search returns a 5xx or 408 (Request Timeout) response.

Telemetry

Trace with ETW or by registering a custom trace provider. For more information, see the [AutoRest documentation](#).

Service Bus

Service Bus is a cloud messaging platform that provides loosely coupled message exchange with improved scale and resiliency for components of an application, whether hosted in the cloud or on-premises.

Retry mechanism

Service Bus implements retries using implementations of the abstract [RetryPolicy](#) class. The namespace and some of the configuration details depend on which Service Bus client SDK package is used:

PACKAGE	DESCRIPTION	NAMESPACE
Microsoft.Azure.ServiceBus	Azure Service Bus client library for .NET Standard.	Microsoft.ServiceBus
WindowsAzure.ServiceBus	This package is the older Service Bus client library. It requires .NET Framework 4.5.2.	Microsoft.Azure.ServiceBus

Both versions of the client library provide the following built-in implementations of `RetryPolicy`:

- `RetryExponential`. Implements exponential backoff.
- `NoRetry`. Does not perform retries. Use this class when you don't need retries at the Service Bus API level, for example when another process manages retries as part of a batch or multistep operation.

The `RetryPolicy.Default` property returns a default policy of type `RetryExponential`. This policy object has the following settings:

SETTING	DEFAULT VALUE	MEANING
MinimalBackoff	0	Minimum back-off interval. Added to the retry interval computed from <code>deltaBackoff</code> .
MaximumBackoff	30 seconds	Maximum back-off interval.
DeltaBackoff	3 seconds	Back-off interval between retries. Multiples of this timespan are used for subsequent retry attempts.
MaxRetryCount	5	The maximum number of retries. (Default value is 10 in the <code>WindowsAzure.ServiceBus</code> package.)

In addition, the following property is defined in the older `WindowsAzure.ServiceBus` package:

SETTING	DEFAULT VALUE	MEANING
TerminationTimeBuffer	5 seconds	Retry attempts will be abandoned if the remaining time is less than this value.

Service Bus actions can return a range of exceptions, listed in [Service Bus messaging exceptions](#). Exceptions returned from Service Bus expose the `IsTransient` property that indicates whether the client should retry the operation. The built-in `RetryExponential` policy checks this property before retrying.

If the last exception encountered was `ServerBusyException`, the `RetryExponential` policy adds 10 seconds to the computed retry interval. This value cannot be changed.

Custom implementations could use a combination of the exception type and the `IsTransient` property to provide more fine-grained control over retry actions. For example, you could detect a `QuotaExceededException` and take action to drain the queue before retrying sending a message to it.

The following code sets the retry policy on a Service Bus client using the `Microsoft.Azure.ServiceBus` library:

```

const string QueueName = "queue1";
const string ServiceBusConnectionString = "<your_connection_string>";

var policy = new RetryExponential(
    minimumBackoff: TimeSpan.FromSeconds(10),
    maximumBackoff: TimeSpan.FromSeconds(30),
    maximumRetryCount: 3);
var queueClient = new QueueClient(ServiceBusConnectionString, QueueName, ReceiveMode.PeekLock, policy);

```

The retry policy cannot be set at the individual operation level. It applies to all operations for the client.

Retry usage guidance

Consider the following guidelines when using Service Bus:

- When using the built-in **RetryExponential** implementation, do not implement a fallback operation as the policy reacts to Server Busy exceptions and automatically switches to an appropriate retry mode.
- Service Bus supports a feature called Paired Namespaces that implements automatic failover to a backup queue in a separate namespace if the queue in the primary namespace fails. Messages from the secondary queue can be sent back to the primary queue when it recovers. This feature helps to address transient failures. For more information, see [Asynchronous Messaging Patterns and High Availability](#).

Consider starting with the following settings for retrying operations. These settings are general purpose, and you should monitor the operations and fine-tune the values to suit your own scenario.

CONTEXT	EXAMPLE MAXIMUM LATENCY	RETRY POLICY	SETTINGS	HOW IT WORKS
Interactive, UI, or foreground	2 seconds*	Exponential	MinimumBackoff = 0 MaximumBackoff = 30 sec. DeltaBackoff = 300 msec. TimeBuffer = 300 msec. MaxRetryCount = 2	Attempt 1: Delay 0 sec. Attempt 2: Delay ~300 msec. Attempt 3: Delay ~900 msec.
Background or batch	30 seconds	Exponential	MinimumBackoff = 1 MaximumBackoff = 30 sec. DeltaBackoff = 1.75 sec. TimeBuffer = 5 sec. MaxRetryCount = 3	Attempt 1: Delay ~1 sec. Attempt 2: Delay ~3 sec. Attempt 3: Delay ~6 sec. Attempt 4: Delay ~13 sec.

* Not including additional delay that is added if a Server Busy response is received.

Telemetry

Service Bus logs retries as ETW events using an **EventSource**. You must attach an **EventListener** to the event source to capture the events and view them in Performance Viewer, or write them to a suitable destination log. The retry events are of the following form:

```
Microsoft-ServiceBus-Client/RetryPolicyIteration
ThreadID="14,500"
FormattedMessage="[TrackingId:] RetryExponential: Operation Get:https://retry-
tests.servicebus.windows.net/TestQueue/?api-version=2014-05 at iteration 0 is retrying after
00:00:00.1000000 sleep because of Microsoft.ServiceBus.Messaging.MessagingCommunicationException: The remote
name could not be resolved: 'retry-tests.servicebus.windows.net'.TrackingId:6a26f99c-dc6d-422e-8565-
f89fdd0d4fe3, TimeStamp:9/5/2014 10:00:13 PM."
trackingId=""
policyType="RetryExponential"
operation="Get:https://retry-tests.servicebus.windows.net/TestQueue/?api-version=2014-05"
iteration="0"
iterationSleep="00:00:00.1000000"
lastExceptionType="Microsoft.ServiceBus.Messaging.MessagingCommunicationException"
exceptionMessage="The remote name could not be resolved: 'retry-
tests.servicebus.windows.net'.TrackingId:6a26f99c-dc6d-422e-8565-f89fdd0d4fe3,TimeStamp:9/5/2014 10:00:13
PM"
```

Examples

The following code example shows how to set the retry policy for:

- A namespace manager. The policy applies to all operations on that manager, and cannot be overridden for individual operations.
- A messaging factory. The policy applies to all clients created from that factory, and cannot be overridden when creating individual clients.
- An individual messaging client. After a client has been created, you can set the retry policy for that client. The policy applies to all operations on that client.

```
using System;
using System.Threading.Tasks;
using Microsoft.ServiceBus;
using Microsoft.ServiceBus.Messaging;

namespace RetryCodeSamples
{
    class ServiceBusCodeSamples
    {
        private const string connectionString =
            @"Endpoint=sb://[my-namespace].servicebus.windows.net/;
SharedAccessKeyName=RootManageSharedAccessKey;
SharedAccessKey=C99.....Mk=";

        public async static Task Samples()
        {
            const string QueueName = "TestQueue";

            ServiceBusEnvironment.SystemConnectivity.Mode = ConnectivityMode.Http;

            var namespaceManager = NamespaceManager.CreateFromConnectionString(connectionString);

            // The namespace manager will have a default exponential policy with 10 retry attempts
            // and a 3 second delay delta.
            // Retry delays will be approximately 0 sec, 3 sec, 9 sec, 25 sec and the fixed 30 sec,
            // with an extra 10 sec added when receiving a ServiceBusyException.

            {
                // Set different values for the retry policy, used for all operations on the namespace
                // manager.
                namespaceManager.Settings.RetryPolicy =
                    new RetryExponential(
                        minBackoff: TimeSpan.FromSeconds(0),
                        maxBackoff: TimeSpan.FromSeconds(30),
                        maxRetryCount: 3);
            }
        }
    }
}
```

```

        // Policies cannot be specified on a per-operation basis.
        if (!await namespaceManager.QueueExistsAsync(QueueName))
        {
            await namespaceManager.CreateQueueAsync(QueueName);
        }
    }

    var messagingFactory = MessagingFactory.Create(
        namespaceManager.Address, namespaceManager.Settings.TokenProvider);
    // The messaging factory will have a default exponential policy with 10 retry attempts
    // and a 3 second delay delta.
    // Retry delays will be approximately 0 sec, 3 sec, 9 sec, 25 sec and the fixed 30 sec,
    // with an extra 10 sec added when receiving a ServiceBusyException.

    {
        // Set different values for the retry policy, used for clients created from it.
        messagingFactory.RetryPolicy =
            new RetryExponential(
                minBackoff: TimeSpan.FromSeconds(1),
                maxBackoff: TimeSpan.FromSeconds(30),
                maxRetryCount: 3);

        // Policies cannot be specified on a per-operation basis.
        var session = await messagingFactory.AcceptMessageSessionAsync();
    }

    {
        var client = messagingFactory.CreateQueueClient(QueueName);
        // The client inherits the policy from the factory that created it.

        // Set different values for the retry policy on the client.
        client.RetryPolicy =
            new RetryExponential(
                minBackoff: TimeSpan.FromSeconds(0.1),
                maxBackoff: TimeSpan.FromSeconds(30),
                maxRetryCount: 3);

        // Policies cannot be specified on a per-operation basis.
        var session = await client.AcceptMessageSessionAsync();
    }
}
}

```

More information

- [Asynchronous Messaging Patterns and High Availability](#)

Service Fabric

Distributing reliable services in a Service Fabric cluster guards against most of the potential transient faults discussed in this article. Some transient faults are still possible, however. For example, the naming service might be in the middle of a routing change when it gets a request, causing it to throw an exception. If the same request comes 100 milliseconds later, it will probably succeed.

Internally, Service Fabric manages this kind of transient fault. You can configure some settings by using the `OperationRetrySettings` class while setting up your services. The following code shows an example. In most cases, this should not be necessary, and the default settings will be fine.

```

FabricTransportRemotingSettings transportSettings = new FabricTransportRemotingSettings
{
    OperationTimeout = TimeSpan.FromSeconds(30)
};

var retrySettings = new OperationRetrySettings(TimeSpan.FromSeconds(15), TimeSpan.FromSeconds(1), 5);

var clientFactory = new FabricTransportServiceRemotingClientFactory(transportSettings);

var serviceProxyFactory = new ServiceProxyFactory((c) => clientFactory, retrySettings);

var client = serviceProxyFactory.CreateServiceProxy<ISomeService>(
    new Uri("fabric:/SomeApp/SomeStatefulReliableService"),
    new ServicePartitionKey(0));

```

More information

- [Remote exception handling](#)

SQL Database using ADO.NET

SQL Database is a hosted SQL database available in a range of sizes and as both a standard (shared) and premium (non-shared) service.

Retry mechanism

SQL Database has no built-in support for retries when accessed using ADO.NET. However, the return codes from requests can be used to determine why a request failed. For more information about SQL Database throttling, see [Azure SQL Database resource limits](#). For a list of relevant error codes, see [SQL error codes for SQL Database client applications](#).

You can use the Polly library to implement retries for SQL Database. See [Transient fault handling with Polly](#).

Retry usage guidance

Consider the following guidelines when accessing SQL Database using ADO.NET:

- Choose the appropriate service option (shared or premium). A shared instance may suffer longer than usual connection delays and throttling due to the usage by other tenants of the shared server. If more predictable performance and reliable low latency operations are required, consider choosing the premium option.
- Ensure that you perform retries at the appropriate level or scope to avoid non-idempotent operations causing inconsistency in the data. Ideally, all operations should be idempotent so that they can be repeated without causing inconsistency. Where this is not the case, the retry should be performed at a level or scope that allows all related changes to be undone if one operation fails; for example, from within a transactional scope. For more information, see [Cloud Service Fundamentals Data Access Layer – Transient Fault Handling](#).
- A fixed interval strategy is not recommended for use with Azure SQL Database except for interactive scenarios where there are only a few retries at very short intervals. Instead, consider using an exponential back-off strategy for the majority of scenarios.
- Choose a suitable value for the connection and command timeouts when defining connections. Too short a timeout may result in premature failures of connections when the database is busy. Too long a timeout may prevent the retry logic working correctly by waiting too long before detecting a failed connection. The value of the timeout is a component of the end-to-end latency; it is effectively added to the retry delay specified in the retry policy for every retry attempt.
- Close the connection after a certain number of retries, even when using an exponential back off retry logic, and retry the operation on a new connection. Retrying the same operation multiple times on the same connection can be a factor that contributes to connection problems. For an example of this technique, see [Cloud Service Fundamentals Data Access Layer – Transient Fault Handling](#).
- When connection pooling is in use (the default) there is a chance that the same connection will be chosen

from the pool, even after closing and reopening a connection. If this is the case, a technique to resolve it is to call the **ClearPool** method of the **SqlConnection** class to mark the connection as not reusable. However, you should do this only after several connection attempts have failed, and only when encountering the specific class of transient failures such as SQL timeouts (error code -2) related to faulty connections.

- If the data access code uses transactions initiated as **TransactionScope** instances, the retry logic should reopen the connection and initiate a new transaction scope. For this reason, the retryable code block should encompass the entire scope of the transaction.

Consider starting with the following settings for retrying operations. These settings are general purpose, and you should monitor the operations and fine-tune the values to suit your own scenario.

CONTEXT	SAMPLE TARGET E2E MAX LATENCY	RETRY STRATEGY	SETTINGS	VALUES	HOW IT WORKS
Interactive, UI, or foreground	2 sec	FixedInterval	Retry count Retry interval First fast retry	3 500 ms true	Attempt 1 - delay 0 sec Attempt 2 - delay 500 ms Attempt 3 - delay 500 ms
Background or batch	30 sec	ExponentialBackoff	Retry count Min back-off Max back-off Delta back-off First fast retry	5 0 sec 60 sec 2 sec false	Attempt 1 - delay 0 sec Attempt 2 - delay ~2 sec Attempt 3 - delay ~6 sec Attempt 4 - delay ~14 sec Attempt 5 - delay ~30 sec

NOTE

The end-to-end latency targets assume the default timeout for connections to the service. If you specify longer connection timeouts, the end-to-end latency will be extended by this additional time for every retry attempt.

Examples

This section shows how you can use Polly to access Azure SQL Database using a set of retry policies configured in the `Policy` class.

The following code shows an extension method on the `SqlCommand` class that calls `ExecuteAsync` with exponential backoff.

```

public async static Task<SqlDataReader> ExecuteReaderWithRetryAsync(this SqlCommand command)
{
    GuardConnectionIsNotNull(command);

    var policy = Policy.Handle<Exception>().WaitAndRetryAsync(
        retryCount: 3, // Retry 3 times
        sleepDurationProvider: attempt => TimeSpan.FromMilliseconds(200 * Math.Pow(2, attempt - 1)), //
        Exponential backoff based on an initial 200 ms delay.
        onRetry: (exception, attempt) =>
    {
        // Capture some information for logging/telemetry.
        logger.LogWarning($"ExecuteReaderWithRetryAsync: Retry {attempt} due to {exception}.");
    });

    // Retry the following call according to the policy.
    await policy.ExecuteAsync<SqlDataReader>(async token =>
    {
        // This code is executed within the Policy

        if (conn.State != System.Data.ConnectionState.Open) await conn.OpenAsync(token);
        return await command.ExecuteReaderAsync(System.Data.CommandBehavior.Default, token);

    }, cancellationToken);
}

```

This asynchronous extension method can be used as follows.

```

var sqlCommand = sqlConnection.CreateCommand();
sqlCommand.CommandText = "[some query]";

using (var reader = await sqlCommand.ExecuteReaderWithRetryAsync())
{
    // Do something with the values
}

```

More information

- [Cloud Service Fundamentals Data Access Layer – Transient Fault Handling](#)

For general guidance on getting the most from SQL Database, see [Azure SQL Database performance and elasticity guide](#).

SQL Database using Entity Framework 6

SQL Database is a hosted SQL database available in a range of sizes and as both a standard (shared) and premium (non-shared) service. Entity Framework is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write.

Retry mechanism

Retry support is provided when accessing SQL Database using Entity Framework 6.0 and higher through a mechanism called [Connection resiliency / retry logic](#). The main features of the retry mechanism are:

- The primary abstraction is the **IDbExecutionStrategy** interface. This interface:
 - Defines synchronous and asynchronous **Execute** methods.
 - Defines classes that can be used directly or can be configured on a database context as a default strategy, mapped to provider name, or mapped to a provider name and server name. When configured on a context, retries occur at the level of individual database operations, of which there might be several for a given context operation.

- Defines when to retry a failed connection, and how.
- It includes several built-in implementations of the **IDbExecutionStrategy** interface:
 - Default: no retrying.
 - Default for SQL Database (automatic): no retrying, but inspects exceptions and wraps them with suggestion to use the SQL Database strategy.
 - Default for SQL Database: exponential (inherited from base class) plus SQL Database detection logic.
- It implements an exponential back-off strategy that includes randomization.
- The built-in retry classes are stateful and are not thread-safe. However, they can be reused after the current operation is completed.
- If the specified retry count is exceeded, the results are wrapped in a new exception. It does not bubble up the current exception.

Policy configuration

Retry support is provided when accessing SQL Database using Entity Framework 6.0 and higher. Retry policies are configured programmatically. The configuration cannot be changed on a per-operation basis.

When configuring a strategy on the context as the default, you specify a function that creates a new strategy on demand. The following code shows how you can create a retry configuration class that extends the **DbConfiguration** base class.

```
public class BloggingContextConfiguration : DbConfiguration
{
    public BlogConfiguration()
    {
        // Set up the execution strategy for SQL Database (exponential) with 5 retries and 4 sec delay
        this.SetExecutionStrategy(
            "System.Data.SqlClient", () => new SqlAzureExecutionStrategy(5, TimeSpan.FromSeconds(4)));
    }
}
```

You can then specify this as the default retry strategy for all operations using the **SetConfiguration** method of the **DbConfiguration** instance when the application starts. By default, EF will automatically discover and use the configuration class.

```
DbConfiguration.SetConfiguration(new BloggingContextConfiguration());
```

You can specify the retry configuration class for a context by annotating the context class with a **DbConfigurationType** attribute. However, if you have only one configuration class, EF will use it without the need to annotate the context.

```
[DbConfigurationType(typeof(BloggingContextConfiguration))]
public class BloggingContext : DbContext
```

If you need to use different retry strategies for specific operations, or disable retries for specific operations, you can create a configuration class that allows you to suspend or swap strategies by setting a flag in the **CallContext**. The configuration class can use this flag to switch strategies, or disable the strategy you provide and use a default strategy. For more information, see [Suspend Execution Strategy](#) (EF6 onwards).

Another technique for using specific retry strategies for individual operations is to create an instance of the required strategy class and supply the desired settings through parameters. You then invoke its **ExecuteAsync** method.

```

var executionStrategy = new SqlAzureExecutionStrategy(5, TimeSpan.FromSeconds(4));
var blogs = await executionStrategy.ExecuteAsync(
    async () =>
{
    using (var db = new BloggingContext("Blogs"))
    {
        // Acquire some values asynchronously and return them
    }
},
new CancellationToken()
);

```

The simplest way to use a `DbConfiguration` class is to locate it in the same assembly as the `DbContext` class. However, this is not appropriate when the same context is required in different scenarios, such as different interactive and background retry strategies. If the different contexts execute in separate AppDomains, you can use the built-in support for specifying configuration classes in the configuration file or set it explicitly using code. If the different contexts must execute in the same AppDomain, a custom solution will be required.

For more information, see [Code-Based Configuration](#) (EF6 onwards).

The following table shows the default settings for the built-in retry policy when using EF6.

SETTING	DEFAULT VALUE	MEANING
Policy	Exponential	Exponential back-off
MaxRetryCount	5	The maximum number of retries.
MaxDelay	30 seconds	The maximum delay between retries. This value does not affect how the series of delays are computed. It only defines an upper bound.
DefaultCoefficient	1 second	The coefficient for the exponential back-off computation. This value cannot be changed.
DefaultRandomFactor	1.1	The multiplier used to add a random delay for each entry. This value cannot be changed.
DefaultExponentialBase	2	The multiplier used to calculate the next delay. This value cannot be changed.

Retry usage guidance

Consider the following guidelines when accessing SQL Database using EF6:

- Choose the appropriate service option (shared or premium). A shared instance may suffer longer than usual connection delays and throttling due to the usage by other tenants of the shared server. If predictable performance and reliable low latency operations are required, consider choosing the premium option.
- A fixed interval strategy is not recommended for use with Azure SQL Database. Instead, use an exponential back-off strategy because the service may be overloaded, and longer delays allow more time for it to recover.
- Choose a suitable value for the connection and command timeouts when defining connections. Base the

timeout on both your business logic design and through testing. You may need to modify this value over time as the volumes of data or the business processes change. Too short a timeout may result in premature failures of connections when the database is busy. Too long a timeout may prevent the retry logic working correctly by waiting too long before detecting a failed connection. The value of the timeout is a component of the end-to-end latency, although you cannot easily determine how many commands will execute when saving the context. You can change the default timeout by setting the **CommandTimeout** property of the **DbContext** instance.

- Entity Framework supports retry configurations defined in configuration files. However, for maximum flexibility on Azure you should consider creating the configuration programmatically within the application. The specific parameters for the retry policies, such as the number of retries and the retry intervals, can be stored in the service configuration file and used at runtime to create the appropriate policies. This allows the settings to be changed without requiring the application to be restarted.

Consider starting with the following settings for retrying operations. You cannot specify the delay between retry attempts (it is fixed and generated as an exponential sequence). You can specify only the maximum values, as shown here; unless you create a custom retry strategy. These settings are general purpose, and you should monitor the operations and fine-tune the values to suit your own scenario.

CONTEXT	SAMPLE TARGET E2E MAX LATENCY	RETRY POLICY	SETTINGS	VALUES	HOW IT WORKS
Interactive, UI, or foreground	2 seconds	Exponential	MaxRetryCount MaxDelay	3 750 ms	Attempt 1 - delay 0 sec Attempt 2 - delay 750 ms Attempt 3 – delay 750 ms
Background or batch	30 seconds	Exponential	MaxRetryCount MaxDelay	5 12 seconds	Attempt 1 - delay 0 sec Attempt 2 - delay ~1 sec Attempt 3 - delay ~3 sec Attempt 4 - delay ~7 sec Attempt 5 - delay 12 sec

NOTE

The end-to-end latency targets assume the default timeout for connections to the service. If you specify longer connection timeouts, the end-to-end latency will be extended by this additional time for every retry attempt.

Examples

The following code example defines a simple data access solution that uses Entity Framework. It sets a specific retry strategy by defining an instance of a class named **BlogConfiguration** that extends **DbConfiguration**.

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.SqlServer;
using System.Threading.Tasks;

namespace RetryCodeSamples
{
    public class BlogConfiguration : DbConfiguration
    {
        public BlogConfiguration()
        {
            // Set up the execution strategy for SQL Database (exponential) with 5 retries and 12 sec delay.
            // These values could be loaded from configuration rather than being hard-coded.
            this.SetExecutionStrategy(
                "System.Data.SqlClient", () => new SqlAzureExecutionStrategy(5,
TimeSpan.FromSeconds(12)));
        }
    }

    // Specify the configuration type if more than one has been defined.
    // [DbConfigurationType(typeof(BlogConfiguration))]
    public class BloggingContext : DbContext
    {
        // Definition of content goes here.
    }

    class EF6CodeSamples
    {
        public async static Task Samples()
        {
            // Execution strategy configured by DbConfiguration subclass, discovered automatically or
            // or explicitly indicated through configuration or with an attribute. Default is no retries.
            using (var db = new BloggingContext("Blogs"))
            {
                // Add, edit, delete blog items here, then:
                await db.SaveChangesAsync();
            }
        }
    }
}

```

More examples of using the Entity Framework retry mechanism can be found in [Connection resiliency / retry logic](#).

More information

- [Azure SQL Database performance and elasticity guide](#)

SQL Database using Entity Framework Core

[Entity Framework Core](#) is an object-relational mapper that enables .NET Core developers to work with data using domain-specific objects. It eliminates the need for most of the data-access code that developers usually need to write. This version of Entity Framework was written from the ground up, and doesn't automatically inherit all the features from EF6.x.

Retry mechanism

Retry support is provided when accessing SQL Database using Entity Framework Core through a mechanism called [connection resiliency](#). Connection resiliency was introduced in EF Core 1.1.0.

The primary abstraction is the `IExecutionStrategy` interface. The execution strategy for SQL Server, including SQL Azure, is aware of the exception types that can be retried and has sensible defaults for maximum retries, delay between retries, and so on.

Examples

The following code enables automatic retries when configuring the DbContext object, which represents a session with the database.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(
            @"Server=
(localdb)\mssqllocaldb;Database=EFMiscellaneous.ConnectionResiliency;Trusted_Connection=True;",
            options => options.EnableRetryOnFailure());
}
```

The following code shows how to execute a transaction with automatic retries, by using an execution strategy. The transaction is defined in a delegate. If a transient failure occurs, the execution strategy will invoke the delegate again.

```
using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    strategy.Execute(() =>
    {
        using (var transaction = db.Database.BeginTransaction())
        {
            db.Blogs.Add(new Blog { Url = "https://blogs.msdn.com/dotnet" });
            db.SaveChanges();

            db.Blogs.Add(new Blog { Url = "https://blogs.msdn.com/visualstudio" });
            db.SaveChanges();

            transaction.Commit();
        }
    });
}
```

Azure Storage

Azure Storage services include blob storage, files, and storage queues.

Blobs, Queues and Files

The ClientOptions Class is the base type for all client option types and exposes various common client options like Diagnostics, Retry, Transport. To provide the client configuration options for connecting to Azure Queue, Blob, and File Storage you must use the corresponding derived type. In the next example, you use the QueueClientOptions class (derived from ClientOptions) to configure a client to connect to Azure Queue Service. The Retry property is the set of options that can be specified to influence how retry attempts are made, and how a failure is eligible to be retried.

```

using System;
using System.Threading;
using Azure.Core;
using Azure.Identity;
using Azure.Storage;
using Azure.Storage.Queues;
using Azure.Storage.Queues.Models;

namespace RetryCodeSamples
{
    class AzureStorageCodeSamples {

        public async static Task Samples() {

            // Provide the client configuration options for connecting to Azure Queue Storage
            QueueClientOptions queueClientOptions = new QueueClientOptions()
            {
                Retry = {
                    Delay = TimeSpan.FromSeconds(2),           //The delay between retry attempts for a fixed
approach or the delay on which to base
                                                //calculations for a backoff-based approach
                    MaxRetries = 5,                          //The maximum number of retry attempts before
giving up
                    Mode = RetryMode.Exponential,          //The approach to use for calculating retry delays
                    MaxDelay = TimeSpan.FromSeconds(10)    //The maximum permissible delay between retry
attempts
                },
                GeoRedundantSecondaryUri = new Uri("https://...")
                // If the GeoRedundantSecondaryUri property is set, the secondary Uri will be used for
GET or HEAD requests during retries.
                // If the status of the response from the secondary Uri is a 404, then subsequent
retries for the request will not use the
                // secondary Uri again, as this indicates that the resource may not have propagated
there yet.
                // Otherwise, subsequent retries will alternate back and forth between primary and
secondary Uri.
            };
            Uri queueServiceUri = new Uri("https://storageaccount.queue.core.windows.net/");
            string accountName = "Storage account name";
            string accountKey = "storage account key";

            // Create a client object for the Queue service, including QueueClientOptions.
            QueueServiceClient serviceClient = new QueueServiceClient(queueServiceUri, new
DefaultAzureCredential(), queueClientOptions);

            CancellationTokenSource source = new CancellationTokenSource();
            CancellationToken cancellationToken = source.Token;

            // Return an async collection of queues in the storage account.
            var queues = serviceClient.GetQueuesAsync(QueueTraits.None, null, cancellationToken);
        }
    }
}

```

Table Support

NOTE

WindowsAzure.Storage Nuget Package has been deprecated. For Azure table support, see [Microsoft.Azure.Cosmos.Table Nuget Package](#)

Retry mechanism

Retries occur at the individual REST operation level and are an integral part of the client API implementation. The client storage SDK uses classes that implement the [IExtendedRetryPolicy Interface](#).

The built-in classes provide support for linear (constant delay) and exponential with randomization retry intervals. There is also a no retry policy for use when another process is handling retries at a higher level. However, you can implement your own retry classes if you have specific requirements not provided by the built-in classes.

Alternate retries switch between primary and secondary storage service location if you are using read access geo-redundant storage (RA-GRS) and the result of the request is a retryable error. See [Azure Storage Redundancy Options](#) for more information.

Policy configuration

Retry policies are configured programmatically. A typical procedure is to create and populate a `TableRequestOptions`, `BlobRequestOptions`, `FileRequestOptions`, or `QueueRequestOptions` instance.

```
TableRequestOptions interactiveRequestOption = new TableRequestOptions()
{
    RetryPolicy = new LinearRetry(TimeSpan.FromMilliseconds(500), 3),
    // For Read-access geo-redundant storage, use PrimaryThenSecondary.
    // Otherwise set this to PrimaryOnly.
    LocationMode = LocationMode.PrimaryThenSecondary,
    // Maximum execution time based on the business use case.
    MaximumExecutionTime = TimeSpan.FromSeconds(2)
};
```

The request options instance can then be set on the client, and all operations with the client will use the specified request options.

```
client.DefaultRequestOptions = interactiveRequestOption;
var stats = await client.GetServiceStatsAsync();
```

You can override the client request options by passing a populated instance of the request options class as a parameter to operation methods.

```
var stats = await client.GetServiceStatsAsync(interactiveRequestOption, operationContext: null);
```

You use an `OperationContext` instance to specify the code to execute when a retry occurs and when an operation has completed. This code can collect information about the operation for use in logs and telemetry.

```
// Set up notifications for an operation
var context = new OperationContext();
context.ClientRequestID = "some request id";
context.Retrying += (sender, args) =>
{
    // Collect retry information
};
context.RequestCompleted += (sender, args) =>
{
    // Collect operation completion information
};
var stats = await client.GetServiceStatsAsync(null, context);
```

In addition to indicating whether a failure is suitable for retry, the extended retry policies return a `RetryContext` object that indicates the number of retries, the results of the last request, whether the next retry will happen in the primary or secondary location (see table below for details). The properties of the `RetryContext` object can be used to decide if and when to attempt a retry. For more information, see [IExtendedRetryPolicy.Evaluate Method](#).

The following tables show the default settings for the built-in retry policies.

Request options:

SETTING	DEFAULT VALUE	MEANING
MaximumExecutionTime	None	Maximum execution time for the request, including all potential retry attempts. If it is not specified, then the amount of time that a request is permitted to take is unlimited. In other words, the request might stop responding.
ServerTimeout	None	Server timeout interval for the request (value is rounded to seconds). If not specified, it will use the default value for all requests to the server. Usually, the best option is to omit this setting so that the server default is used.
LocationMode	None	If the storage account is created with the Read access geo-redundant storage (RA-GRS) replication option, you can use the location mode to indicate which location should receive the request. For example, if PrimaryThenSecondary is specified, requests are always sent to the primary location first. If a request fails, it is sent to the secondary location.
RetryPolicy	ExponentialPolicy	See below for details of each option.

Exponential policy:

SETTING	DEFAULT VALUE	MEANING
maxAttempt	3	Number of retry attempts.
deltaBackoff	4 seconds	Back-off interval between retries. Multiples of this timespan, including a random element, will be used for subsequent retry attempts.
MinBackoff	3 seconds	Added to all retry intervals computed from deltaBackoff. This value cannot be changed.
MaxBackoff	120 seconds	MaxBackoff is used if the computed retry interval is greater than MaxBackoff. This value cannot be changed.

Linear policy:

SETTING	DEFAULT VALUE	MEANING
maxAttempt	3	Number of retry attempts.
deltaBackoff	30 seconds	Back-off interval between retries.

Retry usage guidance

Consider the following guidelines when accessing Azure storage services using the storage client API:

- Use the built-in retry policies from the `Microsoft.Azure.Storage.RetryPolicies` namespace where they are appropriate for your requirements. In most cases, these policies will be sufficient.
- Use the **ExponentialRetry** policy in batch operations, background tasks, or non-interactive scenarios. In these scenarios, you can typically allow more time for the service to recover—with a consequently increased chance of the operation eventually succeeding.
- Consider specifying the **MaximumExecutionTime** property of the `RequestOptions` parameter to limit the total execution time, but take into account the type and size of the operation when choosing a timeout value.
- If you need to implement a custom retry, avoid creating wrappers around the storage client classes. Instead, use the capabilities to extend the existing policies through the **IExtendedRetryPolicy** interface.
- If you are using read access geo-redundant storage (RA-GRS) you can use the **LocationMode** to specify that retry attempts will access the secondary read-only copy of the store should the primary access fail. However, when using this option you must ensure that your application can work successfully with data that may be stale if the replication from the primary store has not yet completed.

Consider starting with the following settings for retrying operations. These settings are general purpose, and you should monitor the operations and fine-tune the values to suit your own scenario.

CONTEXT	SAMPLE TARGET E2E MAX LATENCY	RETRY POLICY	SETTINGS	VALUES	HOW IT WORKS
Interactive, UI, or foreground	2 seconds	Linear	maxAttempt deltaBackoff	3 500 ms	Attempt 1 - delay 500 ms Attempt 2 - delay 500 ms Attempt 3 - delay 500 ms
Background or batch	30 seconds	Exponential	maxAttempt deltaBackoff	5 4 seconds	Attempt 1 - delay ~3 sec Attempt 2 - delay ~7 sec Attempt 3 - delay ~15 sec

Telemetry

Retry attempts are logged to a **TraceSource**. You must configure a **TraceListener** to capture the events and write them to a suitable destination log. You can use the **TextWriterTraceListener** or **XmlWriterTraceListener** to write the data to a log file, the **EventLogTraceListener** to write to the Windows Event Log, or the **EventProviderTraceListener** to write trace data to the ETW subsystem. You can also configure autoflushing of the buffer, and the verbosity of events that will be logged (for example, Error, Warning, Informational, and Verbose). For more information, see [Client-side Logging with the .NET Storage Client Library](#).

Operations can receive an `OperationContext` instance, which exposes a `Retrying` event that can be used to attach custom telemetry logic. For more information, see [OperationContext.Retrying Event](#).

Examples

The following code example shows how to create two `TableRequestOptions` instances with different retry settings; one for interactive requests and one for background requests. The example then sets these two retry policies on the client so that they apply for all requests, and also sets the interactive strategy on a specific request so that it overrides the default settings applied to the client.

```
using System;
using System.Threading.Tasks;
using Microsoft.Azure.Cosmos.Table;

namespace RetryCodeSamples
{
    class AzureStorageCodeSamples
    {
        private const string connectionString = "UseDevelopmentStorage=true";

        public async static Task Samples()
        {
            var storageAccount = CloudStorageAccount.Parse(connectionString);

            TableRequestOptions interactiveRequestOption = new TableRequestOptions()
            {
                RetryPolicy = new LinearRetry(TimeSpan.FromMilliseconds(500), 3),
                // For Read-access geo-redundant storage, use PrimaryThenSecondary.
                // Otherwise set this to PrimaryOnly.
                LocationMode = LocationMode.PrimaryThenSecondary,
                // Maximum execution time based on the business use case.
                MaximumExecutionTime = TimeSpan.FromSeconds(2)
            };

            TableRequestOptions backgroundRequestOption = new TableRequestOptions()
            {
                // Client has a default exponential retry policy with 4 sec delay and 3 retry attempts
                // Retry delays will be approximately 3 sec, 7 sec, and 15 sec
                MaximumExecutionTime = TimeSpan.FromSeconds(30),
                // PrimaryThenSecondary in case of Read-access geo-redundant storage, else set this to
                PrimaryOnly
                LocationMode = LocationMode.PrimaryThenSecondary
            };

            var client = storageAccount.CreateCloudTableClient();
            // Client has a default exponential retry policy with 4 sec delay and 3 retry attempts
            // Retry delays will be approximately 3 sec, 7 sec, and 15 sec
            // ServerTimeout and MaximumExecutionTime are not set

            {
                // Set properties for the client (used on all requests unless overridden)
                // Different exponential policy parameters for background scenarios
                client.DefaultRequestOptions = backgroundRequestOption;
                // Linear policy for interactive scenarios
                client.DefaultRequestOptions = interactiveRequestOption;
            }

            {
                // set properties for a specific request
                var stats = await client.GetServiceStatsAsync(interactiveRequestOption, operationContext:
null);
            }

            {
                // Set up notifications for an operation
                var context = new OperationContext();
                context.ClientRequestID = "some request id";
            }
        }
    }
}
```

```

        context.Retrying += (sender, args) =>
    {
        // Collect retry information
    };
    context.RequestCompleted += (sender, args) =>
    {
        // Collect operation completion information
    };
    var stats = await client.GetServiceStatsAsync(null, context);
}
}
}
}

```

More information

- [Azure Storage client Library retry policy recommendations](#)
- [Storage Client Library 2.0 – Implementing retry policies](#)

General REST and retry guidelines

Consider the following when accessing Azure or third-party services:

- Use a systematic approach to managing retries, perhaps as reusable code, so that you can apply a consistent methodology across all clients and all solutions.
- Consider using a retry framework such as [Polly](#) to manage retries if the target service or client has no built-in retry mechanism. This will help you implement a consistent retry behavior, and it may provide a suitable default retry strategy for the target service. However, you may need to create custom retry code for services that have nonstandard behavior, that do not rely on exceptions to indicate transient failures, or if you want to use a **Retry-Response** reply to manage retry behavior.
- The transient detection logic will depend on the actual client API you use to invoke the REST calls. Some clients, such as the newer **HttpClient** class, will not throw exceptions for completed requests with a non-success HTTP status code.
- The HTTP status code returned from the service can help to indicate whether the failure is transient. You may need to examine the exceptions generated by a client or the retry framework to access the status code or to determine the equivalent exception type. The following HTTP codes typically indicate that a retry is appropriate:
 - 408 Request Timeout
 - 429 Too Many Requests
 - 500 Internal Server Error
 - 502 Bad Gateway
 - 503 Service Unavailable
 - 504 Gateway Timeout
- If you base your retry logic on exceptions, the following typically indicate a transient failure where no connection could be established:
 - **WebExceptionStatus.ConnectionClosed**
 - **WebExceptionStatus.ConnectFailure**
 - **WebExceptionStatus.Timeout**
 - **WebExceptionStatus.RequestCanceled**
- In the case of a service unavailable status, the service might indicate the appropriate delay before retrying in the **Retry-After** response header or a different custom header. Services might also send

additional information as custom headers, or embedded in the content of the response.

- Do not retry for status codes representing client errors (errors in the 4xx range) except for a 408 Request Timeout and 429 Too Many Requests.
- Thoroughly test your retry strategies and mechanisms under a range of conditions, such as different network states and varying system loadings.

Retry strategies

The following are the typical types of retry strategy intervals:

- **Exponential**. A retry policy that performs a specified number of retries, using a randomized exponential back off approach to determine the interval between retries. For example:

```
var random = new Random();

var delta = (int)((Math.Pow(2.0, currentRetryCount) - 1.0) *
    random.Next((int)(this.deltaBackoff.TotalMilliseconds * 0.8),
    (int)(this.deltaBackoff.TotalMilliseconds * 1.2)));
var interval = (int)Math.Min(checked(this.minBackoff.TotalMilliseconds + delta),
    this.maxBackoff.TotalMilliseconds);
retryInterval = TimeSpan.FromMilliseconds(interval);
```

- **Incremental**. A retry strategy with a specified number of retry attempts and an incremental time interval between retries. For example:

```
retryInterval = TimeSpan.FromMilliseconds(this.initialInterval.TotalMilliseconds +
    (this.increment.TotalMilliseconds * currentRetryCount));
```

- **LinearRetry**. A retry policy that performs a specified number of retries, using a specified fixed time interval between retries. For example:

```
retryInterval = this.deltaBackoff;
```

Transient fault handling with Polly

[Polly](#) is a library to programmatically handle retries and [circuit breaker](#) strategies. The Polly project is a member of the [.NET Foundation](#). For services where the client does not natively support retries, Polly is a valid alternative and avoids the need to write custom retry code, which can be hard to implement correctly. Polly also provides a way to trace errors when they occur, so that you can log retries.

More information

- [connection resiliency](#)
- [Data Points - EF Core 1.1](#)

Transient fault handling

3/10/2022 • 19 minutes to read • [Edit Online](#)

All applications that communicate with remote services and resources must be sensitive to transient faults. This is especially the case for applications that run in the cloud, where the nature of the environment and connectivity over the Internet means these types of faults are likely to be encountered more often. Transient faults include the momentary loss of network connectivity to components and services, the temporary unavailability of a service, or timeouts that arise when a service is busy. These faults are often self-correcting, and if the action is repeated after a suitable delay it is likely to succeed.

This document covers general guidance for transient fault handling. For information about handling transient faults when using Microsoft Azure services, see [Azure service-specific retry guidelines](#).

Why do transient faults occur in the cloud?

Transient faults can occur in any environment, on any platform or operating system, and in any kind of application. In solutions that run on local on-premises infrastructure, the performance and availability of the application and its components is typically maintained through expensive and often underused hardware redundancy, and components and resources are located close to each other. While this approach makes a failure less likely, it can still result in transient faults - and even an outage through unforeseen events such as external power supply or network issues, or other disaster scenarios.

Cloud hosting, including private cloud systems, can offer higher overall availability by using shared resources, redundancy, automatic failover, and dynamic resource allocation across many commodity compute nodes. However, the nature of these environments can mean that transient faults are more likely to occur. There are several reasons for this:

- Many resources in a cloud environment are shared, and access to these resources is subject to throttling in order to protect the resource. Some services will refuse connections when the load rises to a specific level, or a maximum throughput rate is reached, in order to allow processing of existing requests and to maintain performance of the service for all users. Throttling helps to maintain the quality of service for neighbors and other tenants using the shared resource.
- Cloud environments are built using vast numbers of commodity hardware units. They deliver performance by dynamically distributing the load across multiple computing units and infrastructure components, and deliver reliability by automatically recycling or replacing failed units. This dynamic nature means that transient faults and temporary connection failures may occasionally occur.
- There are often more hardware components, including network infrastructure such as routers and load balancers, between the application and the resources and services it uses. This additional infrastructure can occasionally introduce additional connection latency and transient connection faults.
- Network conditions between the client and the server may be variable, especially when communication crosses the Internet. Even in on-premises locations, heavy traffic loads may slow communication and cause intermittent connection failures.

Challenges

Transient faults can have a huge effect on the perceived availability of an application, even if it has been thoroughly tested under all foreseeable circumstances. To ensure that cloud-hosted applications operate reliably, they must be able to respond to the following challenges:

- The application must be able to detect faults when they occur, and determine if these faults are likely to be transient, more long-lasting, or are terminal failures. Different resources are likely to return different responses when a fault occurs, and these responses may also vary depending on the context of the operation; for example, the response for an error when reading from storage may be different from response for an error when writing to storage. Many resources and services have well-documented transient failure contracts. However, where such information is not available, it may be difficult to discover the nature of the fault and whether it is likely to be transient.
- The application must be able to retry the operation if it determines that the fault is likely to be transient and keep track of the number of times the operation was retried.
- The application must use an appropriate strategy for the retries. This strategy specifies the number of times it should retry, the delay between each attempt, and the actions to take after a failed attempt. The appropriate number of attempts and the delay between each one are often difficult to determine, and vary based on the type of resource as well as the current operating conditions of the resource and the application itself.

General guidelines

The following guidelines will help you to design a suitable transient fault handling mechanism for your applications:

- **Determine if there is a built-in retry mechanism:**
 - Many services provide an SDK or client library that contains a transient fault handling mechanism. The retry policy it uses is typically tailored to the nature and requirements of the target service. Alternatively, REST interfaces for services may return information that is useful in determining whether a retry is appropriate, and how long to wait before the next retry attempt.
 - Use the built-in retry mechanism where available, unless you have specific and well-understood requirements that make a different retry behavior more appropriate.
- **Determine if the operation is suitable for retrying:**
 - You should only retry operations where the faults are transient (typically indicated by the nature of the error), and if there is at least some likelihood that the operation will succeed when reattempted. There is no point in reattempting operations that indicate an invalid operation such as a database update to an item that does not exist, or requests to a service or resource that has suffered a fatal error.
 - In general, you should implement retries only where the full impact of this can be determined, and the conditions are well understood and can be validated. If not, leave it to the calling code to implement retries. Remember that the errors returned from resources and services outside your control may evolve over time, and you may need to revisit your transient fault detection logic.
 - When you create services or components, consider implementing error codes and messages that will help clients determine whether they should retry failed operations. In particular, indicate if the client should retry the operation (perhaps by returning an `isTransient` value) and suggest a suitable delay before the next retry attempt. If you build a web service, consider returning custom errors defined within your service contracts. Even though generic clients may not be able to read these, they will be useful when building custom clients.
- **Determine an appropriate retry count and interval:**
 - It is vital to optimize the retry count and the interval to the type of use case. If you do not retry a sufficient number of times, the application will be unable to complete the operation and is likely to experience a failure. If you retry too many times, or with too short an interval between tries, the

application can potentially hold resources such as threads, connections, and memory for long periods, which will adversely affect the health of the application.

- The appropriate values for the time interval and the number of retry attempts depend on the type of operation being attempted. For example, if the operation is part of a user interaction, the interval should be short and only a few retries attempted to avoid making users wait for a response (which holds open connections and can reduce availability for other users). If the operation is part of a long running or critical workflow, where canceling and restarting the process is expensive or time-consuming, it is appropriate to wait longer between attempts and retry more times.
- Determining the appropriate intervals between retries is the most difficult part of designing a successful strategy. Typical strategies use the following types of retry interval:
 - **Exponential back-off.** The application waits a short time before the first retry, and then exponentially increasing time between each subsequent retry. For example, it may retry the operation after 3 seconds, 12 seconds, 30 seconds, and so on.
 - **Incremental intervals.** The application waits a short time before the first retry, and then incrementally increasing time between each subsequent retry. For example, it may retry the operation after 3 seconds, 7 seconds, 13 seconds, and so on.
 - **Regular intervals.** The application waits for the same period of time between each attempt. For example, it may retry the operation every 3 seconds.
 - **Immediate retry.** Sometimes a transient fault is brief, perhaps due to an event such as a network packet collision or a spike in a hardware component. In this case, retrying the operation immediately is appropriate because it may succeed if the fault has cleared in the time it takes the application to assemble and send the next request. However, there should never be more than one immediate retry attempt, and you should switch to alternative strategies, such as exponential back-off or fallback actions, if the immediate retry fails.
 - **Randomization.** Any of the retry strategies listed above may include a randomization to prevent multiple instances of the client sending subsequent retry attempts at the same time. For example, one instance may retry the operation after 3 seconds, 11 seconds, 28 seconds, and so on, while another instance may retry the operation after 4 seconds, 12 seconds, 26 seconds, and so on. Randomization is a useful technique that may be combined with other strategies.
- As a general guideline, use an exponential back-off strategy for background operations, and immediate or regular interval retry strategies for interactive operations. In both cases, you should choose the delay and the retry count so that the maximum latency for all retry attempts is within the required end-to-end latency requirement.
- Take into account the combination of all the factors that contribute to the overall maximum timeout for a retried operation. These factors include the time taken for a failed connection to produce a response (typically set by a timeout value in the client) as well as the delay between retry attempts and the maximum number of retries. The total of all these times can result in long overall operation times, especially when using an exponential delay strategy where the interval between retries grows rapidly after each failure. If a process must meet a specific service level agreement (SLA), the overall operation time, including all timeouts and delays, must be within the limits defined in the SLA.
- Overly aggressive retry strategies, which have intervals that are too short or retries that are too frequent, can have an adverse effect on the target resource or service. This may prevent the resource or service from recovering from its overloaded state, and it will continue to block or

refuse requests. This results in a vicious circle where more and more requests are sent to the resource or service, and consequently its ability to recover is further reduced.

- Take into account the timeout of the operations when choosing the retry intervals to avoid launching a subsequent attempt immediately (for example, if the timeout period is similar to the retry interval). Also consider if you need to keep the total possible period (the timeout plus the retry intervals) to below a specific total time. Operations that have unusually short or very long timeouts may influence how long to wait, and how often to retry the operation.
- Use the type of the exception and any data it contains, or the error codes and messages returned from the service, to optimize the interval and the number of retries. For example, some exceptions or error codes (such as the HTTP code 503 Service Unavailable with a Retry-After header in the response) may indicate how long the error might last, or that the service has failed and will not respond to any subsequent attempt.

- **Avoid anti-patterns:**

- In the vast majority of cases, you should avoid implementations that include duplicated layers of retry code. Avoid designs that include cascading retry mechanisms, or that implement retry at every stage of an operation that involves a hierarchy of requests, unless you have specific requirements that demand this. In these exceptional circumstances, use policies that prevent excessive numbers of retries and delay periods, and make sure you understand the consequences. For example, if one component makes a request to another, which then accesses the target service, and you implement retry with a count of three on both calls there will be nine retry attempts in total against the service. Many services and resources implement a built-in retry mechanism and you should investigate how you can disable or modify this if you need to implement retries at a higher level.
- Never implement an endless retry mechanism. This is likely to prevent the resource or service recovering from overload situations, and cause throttling and refused connections to continue for a longer period. Use a finite number of retries, or implement a pattern such as [Circuit Breaker](#) to allow the service to recover.
- Never perform an immediate retry more than once.
- Avoid using a regular retry interval, especially when you have a large number of retry attempts, when accessing services and resources in Azure. The optimum approach in this scenario is an exponential back-off strategy with a circuit-breaking capability.
- Prevent multiple instances of the same client, or multiple instances of different clients, from sending retries at the same times. If this is likely to occur, introduce randomization into the retry intervals.

- **Test your retry strategy and implementation:**

- Ensure you fully test your retry strategy implementation under as wide a set of circumstances as possible, especially when both the application and the target resources or services it uses are under extreme load. To check behavior during testing, you can:
 - Inject transient and nontransient faults into the service. For example, send invalid requests or add code that detects test requests and responds with different types of errors. For an example using TestApi, see [Fault Injection Testing with TestApi](#) and [Introduction to TestApi – Part 5: Managed Code Fault Injection APIs](#).
 - Create a mock of the resource or service that returns a range of errors that the real service may return. Ensure you cover all the types of error that your retry strategy is designed to detect.

- Force transient errors to occur by temporarily disabling or overloading the service if it is a custom service that you created and deployed (of course, you should not attempt to overload any shared resources or shared services within Azure).

- For HTTP-based APIs, consider using the FiddlerCore library in your automated tests to change the outcome of HTTP requests, either by adding extra roundtrip times or by changing the response (such as the HTTP status code, headers, body, or other factors). This enables deterministic testing of a subset of the failure conditions, whether transient faults or other types of failure. For more information, see [FiddlerCore](#). For examples of how to use the library, particularly the `HttpMangler` class, examine the [source code for the Azure Storage SDK](#).

- Perform high load factor and concurrent tests to ensure that the retry mechanism and strategy works correctly under these conditions, and does not have an adverse effect on the operation of the client or cause cross-contamination between requests.

- **Manage retry policy configurations:**

- A *retry policy* is a combination of all of the elements of your retry strategy. It defines the detection mechanism that determines whether a fault is likely to be transient, the type of interval to use (such as regular, exponential back-off, and randomization), the actual interval value(s), and the number of times to retry.

- Retries must be implemented in many places within even the simplest application, and in every layer of more complex applications. Rather than hard-coding the elements of each policy at multiple locations, consider using a central point for storing all the policies. For example, store the values such as the interval and retry count in application configuration files, read them at runtime, and programmatically build the retry policies. This makes it easier to manage the settings, and to modify and fine-tune the values in order to respond to changing requirements and scenarios. However, design the system to store the values rather than rereading a configuration file every time, and ensure suitable defaults are used if the values cannot be obtained from configuration.

- In an Azure Cloud Services application, consider storing the values that are used to build the retry policies at runtime in the service configuration file so that they can be changed without needing to restart the application.

- Take advantage of built-in or default retry strategies available in the client APIs you use, but only where they are appropriate for your scenario. These strategies are typically general purpose. In some scenarios they may be all that is required, but in other scenarios they may not offer the full range of options to suit your specific requirements. You must understand how the settings will affect your application through testing to determine the most appropriate values.

- **Log and track transient and nontransient faults:**

- As part of your retry strategy, include exception handling and other instrumentation that logs when retry attempts are made. While an occasional transient failure and retry are to be expected, and do not indicate a problem, regular and increasing numbers of retries are often an indicator of an issue that may cause a failure, or is currently degrading application performance and availability.

- Log transient faults as Warning entries rather than Error entries so that monitoring systems do not detect them as application errors that may trigger false alerts.

- Consider storing a value in your log entries that indicates if the retries were caused by throttling in the service, or by other types of faults such as connection failures, so that you can differentiate them during analysis of the data. An increase in the number of throttling errors is often an indicator of a design flaw in the application or the need to switch to a premium service that offers

dedicated hardware.

- Consider measuring and logging the overall time taken for operations that include a retry mechanism. This is a good indicator of the overall effect of transient faults on user response times, process latency, and the efficiency of the application use cases. Also log the number of retries occurred in order to understand the factors that contributed to the response time.
 - Consider implementing a telemetry and monitoring system that can raise alerts when the number and rate of failures, the average number of retries, or the overall times taken for operations to succeed, is increasing.
- **Manage operations that continually fail:**
 - There will be circumstances where the operation continues to fail at every attempt, and it is vital to consider how you will handle this situation:
 - Although a retry strategy will define the maximum number of times that an operation should be retried, it does not prevent the application repeating the operation again, with the same number of retries. For example, if an order processing service fails with a fatal error that puts it out of action permanently, the retry strategy may detect a connection timeout and consider it to be a transient fault. The code will retry the operation a specified number of times and then give up. However, when another customer places an order, the operation will be attempted again - even though it is sure to fail every time.
 - To prevent continual retries for operations that continually fail, consider implementing the [Circuit Breaker pattern](#). In this pattern, if the number of failures within a specified time window exceeds the threshold, requests are returned to the caller immediately as errors, without attempting to access the failed resource or service.
 - The application can periodically test the service, on an intermittent basis and with long intervals between requests, to detect when it becomes available. An appropriate interval will depend on the scenario, such as the criticality of the operation and the nature of the service, and might be anything between a few minutes and several hours. At the point where the test succeeds, the application can resume normal operations and pass requests to the newly recovered service.
 - In the meantime, it may be possible to fall back to another instance of the service (perhaps in a different datacenter or application), use a similar service that offers compatible (perhaps simpler) functionality, or perform some alternative operations in the hope that the service will become available soon. For example, it may be appropriate to store requests for the service in a queue or data store and replay them later. Otherwise you might be able to redirect the user to an alternative instance of the application, degrade the performance of the application but still offer acceptable functionality, or just return a message to the user indicating that the application is not available at present.

- **Other considerations**

- When deciding on the values for the number of retries and the retry intervals for a policy, consider if the operation on the service or resource is part of a long-running or multistep operation. It may be difficult or expensive to compensate all the other operational steps that have already succeeded when one fails. In this case, a very long interval and a large number of retries may be acceptable as long as it does not block other operations by holding or locking scarce resources.
- Consider if retrying the same operation may cause inconsistencies in data. If some parts of a multistep process are repeated, and the operations are not idempotent, it may result in an inconsistency. For example, an operation that increments a value, if repeated, will produce an invalid result. Repeating an operation that sends a message to a queue may cause an inconsistency

in the message consumer if it cannot detect duplicate messages. To prevent this, ensure that you design each step as an idempotent operation. For more information about idempotency, see [Idempotency patterns](#).

- Consider the scope of the operations that will be retried. For example, it may be easier to implement retry code at a level that encompasses several operations, and retry them all if one fails. However, doing this may result in idempotency issues or unnecessary rollback operations.
- If you choose a retry scope that encompasses several operations, take into account the total latency of all of them when determining the retry intervals, when monitoring the time taken, and before raising alerts for failures.
- Consider how your retry strategy may affect neighbors and other tenants in a shared application, or when using shared resources and services. Aggressive retry policies can cause an increasing number of transient faults to occur for these other users and for applications that share the resources and services. Likewise, your application may be affected by the retry policies implemented by other users of the resources and services. For mission-critical applications, you may decide to use premium services that are not shared. This provides you with much more control over the load and consequent throttling of these resources and services, which can help to justify the additional cost.

More information

- [Azure service-specific retry guidelines](#)
- [Circuit Breaker pattern](#)
- [Compensating Transaction pattern](#)
- [Idempotency patterns](#)

Performance tuning a distributed application

3/10/2022 • 2 minutes to read • [Edit Online](#)

In this series, we walk through several cloud application scenarios, showing how a development team used load tests and metrics to diagnose performance issues. These articles are based on actual load testing that we performed when developing example applications. The code for each scenario is available on GitHub.

Scenarios:

- [Distributed business transaction](#)
- [Calling multiple backend services](#)
- [Event stream processing](#)

What is performance?

Performance is frequently measured in terms of throughput, response time, and availability. Performance targets should be based on business operations. Customer-facing tasks may have more stringent requirements than operational tasks such as generating reports.

Define a service level objective (SLO) that defines performance targets for each workload. You typically achieve this by breaking a performance target into a set of Key Performance Indicators (KPIs), such as:

- Latency or response time of specific requests
- The number of requests performed per second
- The rate at which the system generates exceptions.

Performance targets should explicitly include a target load. Also, not all users will receive exactly the same level of performance, even when accessing the system simultaneously and performing the same work. So an SLO should be framed in terms of percentiles.

An example SLO for might be: "Client requests will have a response within 500 ms @ P90, at loads up to 25 K requests/second."

Challenges of performance tuning a distributed system

It can be especially challenging to diagnose performance issues in a distributed application. Some of the challenges are:

- A single business transaction or operation typically involves multiple components of the system. It can be hard to get a holistic end-to-end view of a single operation.
- Resource consumption is distributed across multiple nodes. To get a consistent view, you need to aggregate logs and metrics in one place.
- The cloud offers elastic scale. Autoscaling is an important technique for handling spikes in load, but it can also mask underlying issues. Also, it can be hard to know which components need to scale and when.
- Cascading failures can cause failures upstream of the root problem. As a result, the first signal of the problem may appear in a different component than the root cause.

General best practices

Performance tuning is both an art and a science, but it can be made closer to science by taking a systematic

approach. Here are some best practices:

- Enable telemetry to collect metrics. Instrument your code. Follow [best practices for monitoring](#). Use correlated tracing so that you can view all the steps in a transaction.
- Monitor the 90/95/99 percentiles, not just average. The average can mask outliers. The sampling rate for metrics also matters. If the sampling rate is too low, it can hide spikes or outliers that might indicate problems.
- Attack one bottleneck at a time. Form a hypothesis and test it by changing one variable at a time. Removing one bottleneck will often uncover another bottleneck further upstream or downstream.
- Errors and retries can have a large impact on performance. If you see that you are being throttled by backend services, scale out or try to optimize usage (for example by tuning database queries).
- Look for common [performance anti-patterns](#).
- Look for opportunities to parallelize. Two common sources of bottlenecks are message queues and databases. In both cases, sharding can help. For more information, see [Horizontal, vertical, and functional data partitioning](#). Look for hot partitions that might indicate imbalanced read or write loads.

Next steps

Read the performance tuning scenarios

- [Distributed business transaction](#)
- [Calling multiple backend services](#)
- [Event stream processing](#)

Performance testing and antipatterns for cloud applications

3/10/2022 • 2 minutes to read • [Edit Online](#)

Performance antipatterns, much like design patterns, are common defective processes and implementations within organizations. These are common practices that are likely to cause scalability problems when an application is under pressure. Awareness of these practices can help simplify communication of high-level concepts amongst software practitioners.

Here is a common scenario: An application behaves well during performance testing. It's released to production, and begins to handle real workloads. At that point, it starts to perform poorly—rejecting user requests, stalling, or throwing exceptions. The development team is then faced with two questions:

- Why didn't this behavior show up during testing?
- How do we fix it?

The answer to the first question is straightforward. It's difficult to simulate real users in a test environment, along with their behavior patterns and the volumes of work they might perform. The only completely sure way to understand how a system behaves under load is to observe it in production. To be clear, we aren't suggesting that you should skip performance testing. Performance testing is crucial for getting baseline performance metrics. But you must be prepared to observe and correct performance issues when they arise in the live system.

The answer to the second question, how to fix the problem, is less straightforward. Any number of factors might contribute, and sometimes the problem only manifests under certain circumstances. Instrumentation and logging are key to finding the root cause, but you also have to know what to look for.

Based on our engagements with Microsoft Azure customers, we've identified some of the most common performance issues that customers see in production. For each antipattern, we describe why the antipattern typically occurs, symptoms of the antipattern, and techniques for resolving the problem. We also provide sample code that illustrates both the antipattern and a suggested scalability solution.

Some of these antipatterns may seem obvious when you read the descriptions, but they occur more often than you might think. Sometimes an application inherits a design that worked on-premises, but doesn't scale in the cloud. Or an application might start with a very clean design, but as new features are added, one or more of these antipatterns creeps in. Regardless, this guide will help you to identify and fix these antipatterns.

Catalog of antipatterns

Here is the list of the antipatterns that we've identified:

ANTIPATTERN	DESCRIPTION
Busy Database	Offloading too much processing to a data store.
Busy Front End	Moving resource-intensive tasks onto background threads.
Chatty I/O	Continually sending many small network requests.

ANTIPATTERN	DESCRIPTION
Extraneous Fetching	Retrieving more data than is needed, resulting in unnecessary I/O.
Improper Instantiation	Repeatedly creating and destroying objects that are designed to be shared and reused.
Monolithic Persistence	Using the same data store for data with very different usage patterns.
No Caching	Failing to cache data.
Noisy Neighbor	A single tenant uses a disproportionate amount of the resources.
Retry Storm	Retrying failed requests to a server too often.
Synchronous I/O	Blocking the calling thread while I/O completes.

Next steps

For more about performance tuning, see [Performance tuning a distributed application](#)

Busy Database antipattern

3/10/2022 • 7 minutes to read • [Edit Online](#)

Offloading processing to a database server can cause it to spend a significant proportion of time running code, rather than responding to requests to store and retrieve data.

Problem description

Many database systems can run code. Examples include stored procedures and triggers. Often, it's more efficient to perform this processing close to the data, rather than transmitting the data to a client application for processing. However, overusing these features can hurt performance, for several reasons:

- The database server may spend too much time processing, rather than accepting new client requests and fetching data.
- A database is usually a shared resource, so it can become a bottleneck during periods of high use.
- Runtime costs may be excessive if the data store is metered. That's particularly true of managed database services. For example, Azure SQL Database charges for [Database Transaction Units](#) (DTUs).
- Databases have finite capacity to scale up, and it's not trivial to scale a database horizontally. Therefore, it may be better to move processing into a compute resource, such as a VM or App Service app, that can easily scale out.

This antipattern typically occurs because:

- The database is viewed as a service rather than a repository. An application might use the database server to format data (for example, converting to XML), manipulate string data, or perform complex calculations.
- Developers try to write queries whose results can be displayed directly to users. For example, a query might combine fields or format dates, times, and currency according to locale.
- Developers are trying to correct the [Extraneous Fetching](#) antipattern by pushing computations to the database.
- Stored procedures are used to encapsulate business logic, perhaps because they are considered easier to maintain and update.

The following example retrieves the 20 most valuable orders for a specified sales territory and formats the results as XML. It uses Transact-SQL functions to parse the data and convert the results to XML. You can find the complete sample [here](#).

```

SELECT TOP 20
    soh.[SalesOrderNumber] AS '@OrderNumber',
    soh.[Status] AS '@Status',
    soh.[ShipDate] AS '@ShipDate',
    YEAR(soh.[OrderDate]) AS '@OrderDateYear',
    MONTH(soh.[OrderDate]) AS '@OrderDateMonth',
    soh.[DueDate] AS '@DueDate',
    FORMAT(ROUND(soh.[SubTotal],2),'C')
        AS '@SubTotal',
    FORMAT(ROUND(soh.[TaxAmt],2),'C')
        AS '@TaxAmt',
    FORMAT(ROUND(soh.[TotalDue],2),'C')
        AS '@TotalDue',
    CASE WHEN soh.[TotalDue] > 5000 THEN 'Y' ELSE 'N' END
        AS '@ReviewRequired',
(
SELECT
    c.[AccountNumber] AS '@AccountNumber',
    UPPER(LTRIM(RTRIM(REPLACE(
        CONCAT( p.[Title], ' ', p.[FirstName], ' ', p.[MiddleName], ' ', p.[LastName], ' ', p.[Suffix]),
        ' ', ' '))) AS '@FullName'
FROM [Sales].[Customer] c
    INNER JOIN [Person].[Person] p
ON c.[PersonID] = p.[BusinessEntityID]
WHERE c.[CustomerID] = soh.[CustomerID]
FOR XML PATH ('Customer'), TYPE
),
(
SELECT
    sod.[OrderQty] AS '@Quantity',
    FORMAT(sod.[UnitPrice],'C')
        AS '@UnitPrice',
    FORMAT(ROUND(sod.[LineTotal],2),'C')
        AS '@LineTotal',
    sod.[ProductID] AS '@ProductId',
    CASE WHEN (sod.[ProductID] >= 710) AND (sod.[ProductID] <= 720) AND (sod.[OrderQty] >= 5) THEN 'Y' ELSE
    'N' END
        AS '@InventoryCheckRequired'

    FROM [Sales].[SalesOrderDetail] sod
    WHERE sod.[SalesOrderID] = soh.[SalesOrderID]
    ORDER BY sod.[SalesOrderDetailID]
    FOR XML PATH ('LineItem'), TYPE, ROOT('OrderLineItems')
)
FROM [Sales].[SalesOrderHeader] soh
WHERE soh.[TerritoryId] = @TerritoryId
ORDER BY soh.[TotalDue] DESC
FOR XML PATH ('Order'), ROOT('Orders')

```

Clearly, this is complex query. As we'll see later, it turns out to use significant processing resources on the database server.

How to fix the problem

Move processing from the database server into other application tiers. Ideally, you should limit the database to performing data access operations, using only the capabilities that the database is optimized for, such as aggregation in an RDBMS.

For example, the previous Transact-SQL code can be replaced with a statement that simply retrieves the data to be processed.

```

SELECT
soh.[SalesOrderNumber] AS [OrderNumber],
soh.[Status] AS [Status],
soh.[OrderDate] AS [OrderDate],
soh.[DueDate] AS [DueDate],
soh.[ShipDate] AS [ShipDate],
soh.[SubTotal] AS [SubTotal],
soh.[TaxAmt] AS [TaxAmt],
soh.[TotalDue] AS [TotalDue],
c.[AccountNumber] AS [AccountNumber],
p.[Title] AS [CustomerTitle],
p.[FirstName] AS [CustomerFirstName],
p.[MiddleName] AS [CustomerMiddleName],
p.[LastName] AS [CustomerLastName],
p.[Suffix] AS [CustomerSuffix],
sod.[OrderQty] AS [Quantity],
sod.[UnitPrice] AS [UnitPrice],
sod.[LineTotal] AS [LineTotal],
sod.[ProductID] AS [ProductId]
FROM [Sales].[SalesOrderHeader] soh
INNER JOIN [Sales].[Customer] c ON soh.[CustomerID] = c.[CustomerID]
INNER JOIN [Person].[Person] p ON c.[PersonID] = p.[BusinessEntityID]
INNER JOIN [Sales].[SalesOrderDetail] sod ON soh.[SalesOrderID] = sod.[SalesOrderID]
WHERE soh.[TerritoryId] = @TerritoryId
AND soh.[SalesOrderId] IN (
    SELECT TOP 20 SalesOrderId
    FROM [Sales].[SalesOrderHeader] soh
    WHERE soh.[TerritoryId] = @TerritoryId
    ORDER BY soh.[TotalDue] DESC)
ORDER BY soh.[TotalDue] DESC, sod.[SalesOrderDetailID]

```

The application then uses the .NET Framework `System.Xml.Linq` APIs to format the results as XML.

```

// Create a new SqlCommand to run the Transact-SQL query
using (var command = new SqlCommand(...))
{
    command.Parameters.AddWithValue("@TerritoryId", id);

    // Run the query and create the initial XML document
    using (var reader = await command.ExecuteReaderAsync())
    {
        var lastOrderNumber = string.Empty;
        var doc = new XDocument();
        var orders = new XElement("Orders");
        doc.Add(orders);

        XElement lineItems = null;
        // Fetch each row in turn, format the results as XML, and add them to the XML document
        while (await reader.ReadAsync())
        {
            var orderNumber = reader["OrderNumber"].ToString();
            if (orderNumber != lastOrderNumber)
            {
                lastOrderNumber = orderNumber;

                var order = new XElement("Order");
                orders.Add(order);
                var customer = new XElement("Customer");
                lineItems = new XElement("OrderLineItems");
                order.Add(customer, lineItems);

                var orderDate = (DateTime)reader["OrderDate"];
                var totalDue = (Decimal)reader["TotalDue"];
                var reviewRequired = totalDue > 5000 ? 'Y' : 'N';

                order.Add(

```

```

        new XAttribute("OrderNumber", orderNumber),
        new XAttribute("Status", reader["Status"]),
        new XAttribute("ShipDate", reader["ShipDate"]),
        ... // More attributes, not shown.

        var fullName = string.Join(" ",
            reader["CustomerTitle"],
            reader["CustomerFirstName"],
            reader["CustomerMiddleName"],
            reader["CustomerLastName"],
            reader["CustomerSuffix"]
        )
        .Replace(" ", " ") //remove double spaces
        .Trim()
        .ToUpper();

        customer.Add(
            new XAttribute("AccountNumber", reader["AccountNumber"]),
            new XAttribute("FullName", fullName));
    }

    var productId = (int)reader["ProductID"];
    var quantity = (short)reader["Quantity"];
    var inventoryCheckRequired = (productId >= 710 && productId <= 720 && quantity >= 5) ? 'Y' :
    'N';

    lineItems.Add(
        new XElement("LineItem",
            new XAttribute("Quantity", quantity),
            new XAttribute("UnitPrice", ((Decimal)reader["UnitPrice"]).ToString("C")),
            new XAttribute("LineTotal", RoundAndFormat(reader["LineTotal"])),
            new XAttribute("ProductId", productId),
            new XAttribute("InventoryCheckRequired", inventoryCheckRequired)
        ));
}
// Match the exact formatting of the XML returned from SQL
var xml = doc
    .ToString(SaveOptions.DisableFormatting)
    .Replace(" />", "/>");
}
}

```

NOTE

This code is somewhat complex. For a new application, you might prefer to use a serialization library. However, the assumption here is that the development team is refactoring an existing application, so the method needs to return the exact same format as the original code.

Considerations

- Many database systems are highly optimized to perform certain types of data processing, such as calculating aggregate values over large datasets. Don't move those types of processing out of the database.
- Do not relocate processing if doing so causes the database to transfer far more data over the network. See the [Extraneous Fetching antipattern](#).
- If you move processing to an application tier, that tier may need to scale out to handle the additional work.

How to detect the problem

Symptoms of a busy database include a disproportionate decline in throughput and response times in operations that access the database.

You can perform the following steps to help identify this problem:

1. Use performance monitoring to identify how much time the production system spends performing database activity.
2. Examine the work performed by the database during these periods.
3. If you suspect that particular operations might cause too much database activity, perform load testing in a controlled environment. Each test should run a mixture of the suspect operations with a variable user load. Examine the telemetry from the load tests to observe how the database is used.
4. If the database activity reveals significant processing but little data traffic, review the source code to determine whether the processing can better be performed elsewhere.

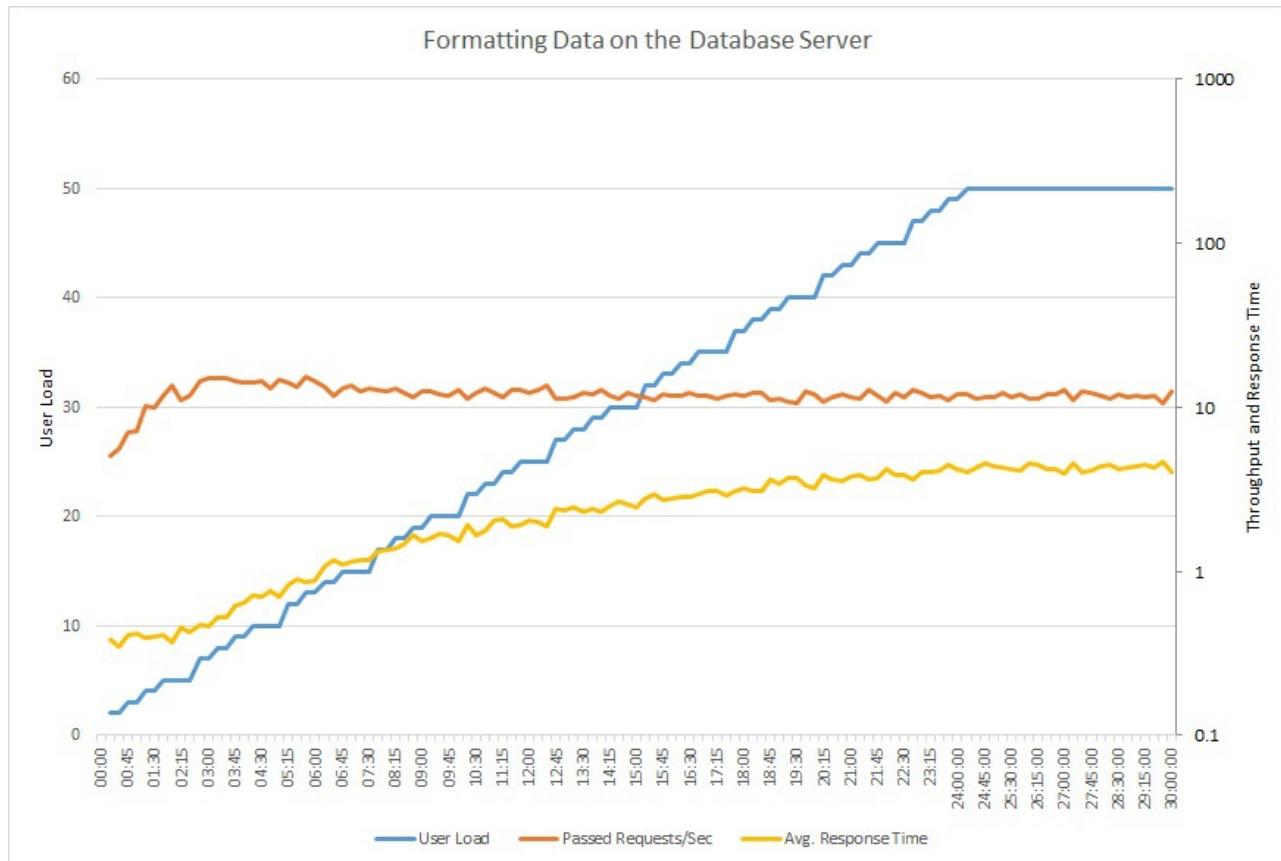
If the volume of database activity is low or response times are relatively fast, then a busy database is unlikely to be a performance problem.

Example diagnosis

The following sections apply these steps to the sample application described earlier.

Monitor the volume of database activity

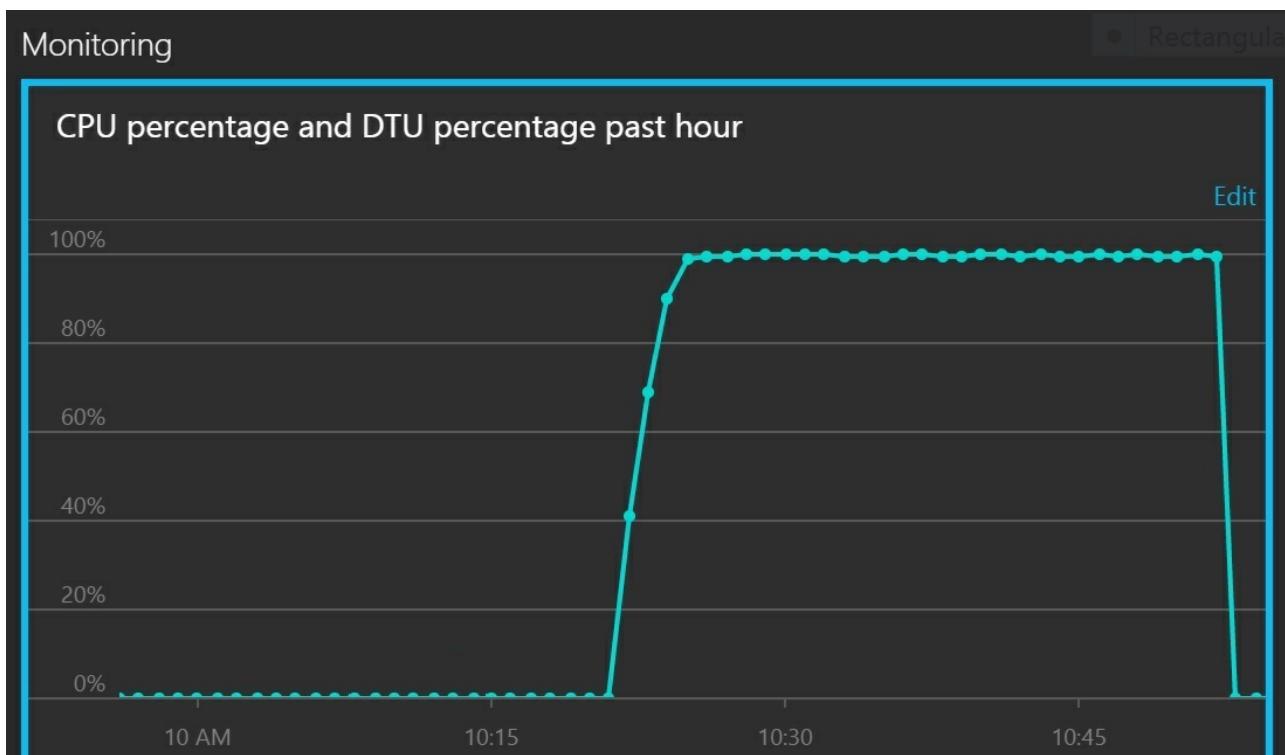
The following graph shows the results of running a load test against the sample application, using a step load of up to 50 concurrent users. The volume of requests quickly reaches a limit and stays at that level, while the average response time steadily increases. A logarithmic scale is used for those two metrics.



This line graph shows user load, requests per second, and average response time. The graph shows that response time increases as load increases.

The next graph shows CPU utilization and DTUs as a percentage of service quota. DTUs provide a measure of how much processing the database performs. The graph shows that CPU and DTU utilization both quickly

reached 100%.



This line graph shows CPU percentage and DTU percentage over time. The graph shows that both quickly reach 100%.

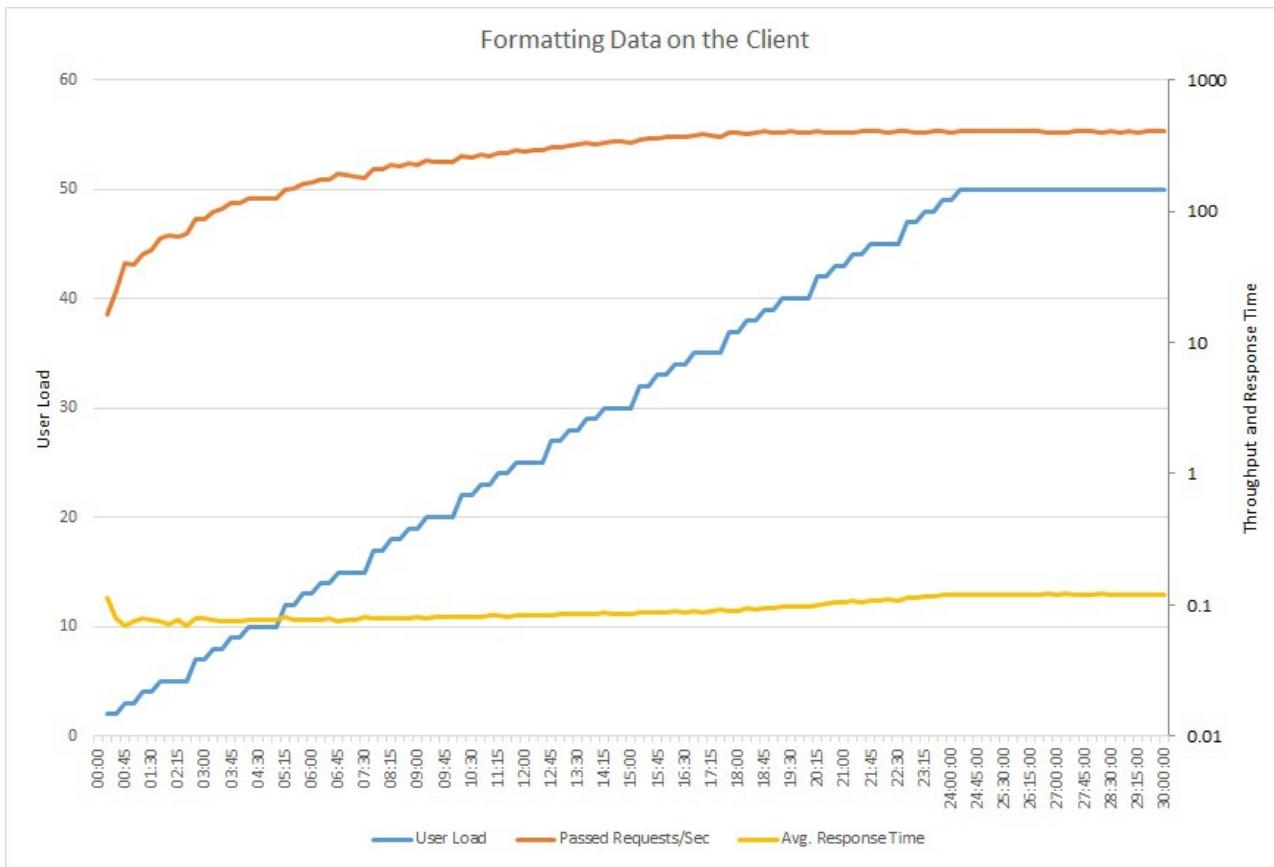
Examine the work performed by the database

It could be that the tasks performed by the database are genuine data access operations, rather than processing, so it is important to understand the SQL statements being run while the database is busy. Monitor the system to capture the SQL traffic and correlate the SQL operations with application requests.

If the database operations are purely data access operations, without a lot of processing, then the problem might be [Extraneous Fetching](#).

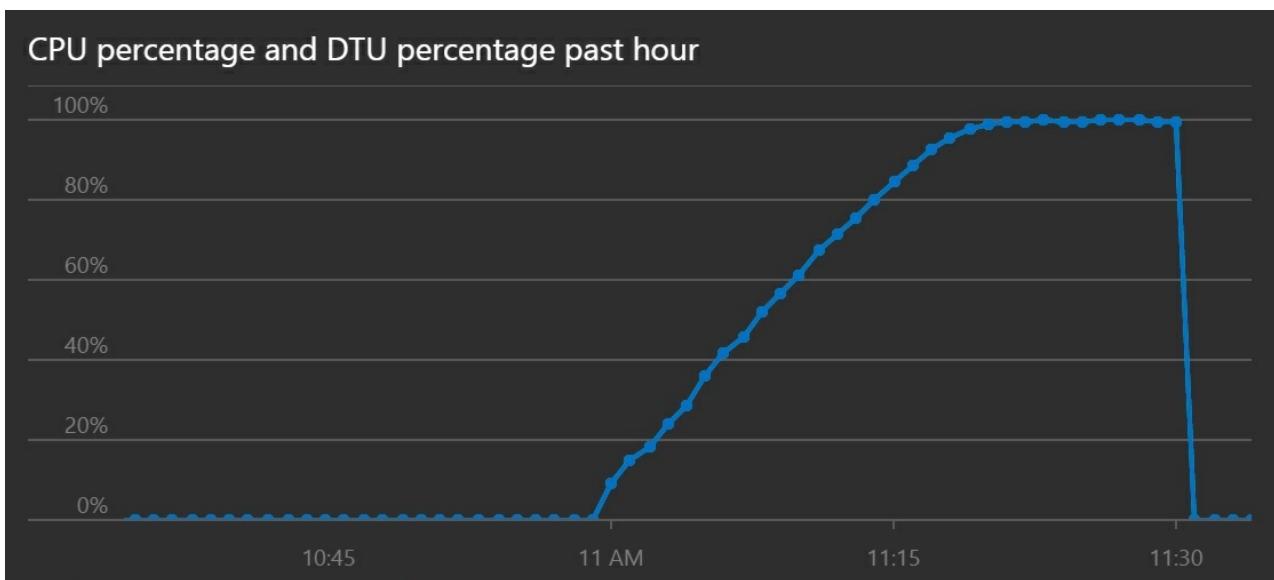
Implement the solution and verify the result

The following graph shows a load test using the updated code. Throughput is significantly higher, over 400 requests per second versus 12 earlier. The average response time is also much lower, just above 0.1 seconds compared to over 4 seconds.



This line graph shows user load, requests per second, and average response time. The graph shows that response time remains roughly constant throughout the load test.

CPU and DTU utilization shows that the system took longer to reach saturation, despite the increased throughput.



This line graph shows CPU percentage and DTU percentage over time. The graph shows that CPU and DTU take longer to reach 100% than previously.

Related resources

- [Extraneous Fetching antipattern](#)

Busy Front End antipattern

3/10/2022 • 8 minutes to read • [Edit Online](#)

Performing asynchronous work on a large number of background threads can starve other concurrent foreground tasks of resources, decreasing response times to unacceptable levels.

Problem description

Resource-intensive tasks can increase the response times for user requests and cause high latency. One way to improve response times is to offload a resource-intensive task to a separate thread. This approach lets the application stay responsive while processing happens in the background. However, tasks that run on a background thread still consume resources. If there are too many of them, they can starve the threads that are handling requests.

NOTE

The term *resource* can encompass many things, such as CPU utilization, memory occupancy, and network or disk I/O.

This problem typically occurs when an application is developed as monolithic piece of code, with all of the business logic combined into a single tier shared with the presentation layer.

Here's an example using ASP.NET that demonstrates the problem. You can find the complete sample [here](#).

```
public class WorkInFrontEndController : ApiController
{
    [HttpPost]
    [Route("api/workinfrontend")]
    public HttpResponseMessage Post()
    {
        new Thread(() =>
        {
            //Simulate processing
            Thread.Sleep(Int32.MaxValue / 100);
        }).Start();

        return Request.CreateResponse(HttpStatusCode.Accepted);
    }
}

public class UserProfileController : ApiController
{
    [HttpGet]
    [Route("api/userprofile/{id}")]
    public UserProfile Get(int id)
    {
        //Simulate processing
        return new UserProfile() { FirstName = "Alton", LastName = "Hudgens" };
    }
}
```

- The `Post` method in the `WorkInFrontEnd` controller implements an HTTP POST operation. This operation simulates a long-running, CPU-intensive task. The work is performed on a separate thread, in an attempt to enable the POST operation to complete quickly.
- The `Get` method in the `UserProfile` controller implements an HTTP GET operation. This method is much

less CPU intensive.

The primary concern is the resource requirements of the `Post` method. Although it puts the work onto a background thread, the work can still consume considerable CPU resources. These resources are shared with other operations being performed by other concurrent users. If a moderate number of users send this request at the same time, overall performance is likely to suffer, slowing down all operations. Users might experience significant latency in the `Get` method, for example.

How to fix the problem

Move processes that consume significant resources to a separate back end.

With this approach, the front end puts resource-intensive tasks onto a message queue. The back end picks up the tasks for asynchronous processing. The queue also acts as a load leveler, buffering requests for the back end. If the queue length becomes too long, you can configure autoscaling to scale out the back end.

Here is a revised version of the previous code. In this version, the `Post` method puts a message on a Service Bus queue.

```
public class WorkInBackgroundController : ApiController
{
    private static readonly QueueClient QueueClient;
    private static readonly string QueueName;
    private static readonly ServiceBusQueueHandler ServiceBusQueueHandler;

    public WorkInBackgroundController()
    {
        var serviceBusConnectionString = ...;
        QueueName = ...;
        ServiceBusQueueHandler = new ServiceBusQueueHandler(serviceBusConnectionString);
        QueueClient = ServiceBusQueueHandler.GetQueueClientAsync(QueueName).Result;
    }

    [HttpPost]
    [Route("api/workinbackground")]
    public async Task<long> Post()
    {
        return await ServiceBusQueueHandler.AddWorkLoadToQueueAsync(QueueClient, QueueName, 0);
    }
}
```

The back end pulls messages from the Service Bus queue and does the processing.

```

public async Task RunAsync(CancellationToken cancellationToken)
{
    this._queueClient.OnMessageAsync(
        // This lambda is invoked for each message received.
        async (receivedMessage) =>
    {
        try
        {
            // Simulate processing of message
            Thread.Sleep(Int32.MaxValue / 1000);

            await receivedMessage.CompleteAsync();
        }
        catch
        {
            receivedMessage.Abandon();
        }
    });
}

```

Considerations

- This approach adds some additional complexity to the application. You must handle queuing and dequeuing safely to avoid losing requests in the event of a failure.
- The application takes a dependency on an additional service for the message queue.
- The processing environment must be sufficiently scalable to handle the expected workload and meet the required throughput targets.
- While this approach should improve overall responsiveness, the tasks that are moved to the back end may take longer to complete.

How to detect the problem

Symptoms of a busy front end include high latency when resource-intensive tasks are being performed. End users are likely to report extended response times or failures caused by services timing out. These failures could also return HTTP 500 (Internal Server) errors or HTTP 503 (Service Unavailable) errors. Examine the event logs for the web server, which are likely to contain more detailed information about the causes and circumstances of the errors.

You can perform the following steps to help identify this problem:

1. Perform process monitoring of the production system, to identify points when response times slow down.
2. Examine the telemetry data captured at these points to determine the mix of operations being performed and the resources being used.
3. Find any correlations between poor response times and the volumes and combinations of operations that were happening at those times.
4. Load test each suspected operation to identify which operations are consuming resources and starving other operations.
5. Review the source code for those operations to determine why they might cause excessive resource consumption.

Example diagnosis

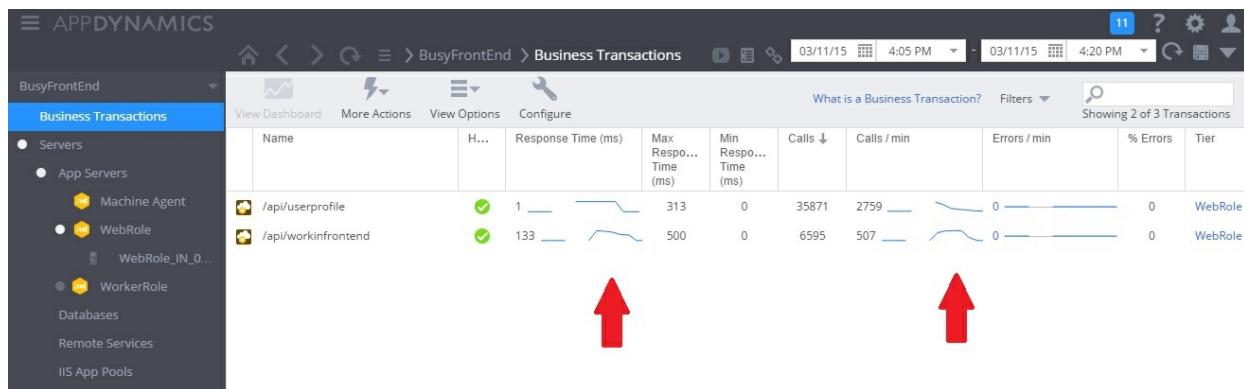
The following sections apply these steps to the sample application described earlier.

Identify points of slowdown

Instrument each method to track the duration and resources consumed by each request. Then monitor the

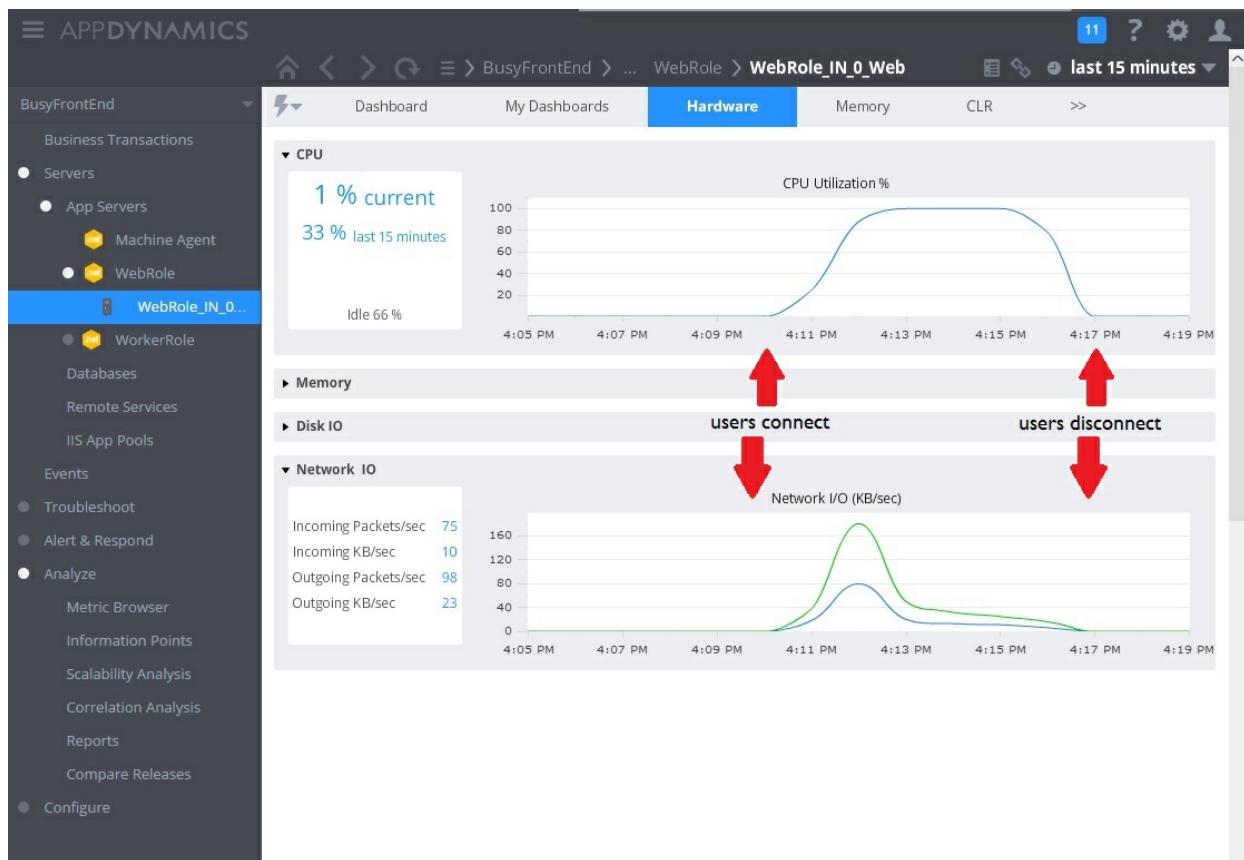
application in production. This can provide an overall view of how requests compete with each other. During periods of stress, slow-running resource-hungry requests will likely affect other operations, and this behavior can be observed by monitoring the system and noting the drop off in performance.

The following image shows a monitoring dashboard. (We used AppDynamics for our tests.) Initially, the system has light load. Then users start requesting the `UserProfile` GET method. The performance is reasonably good until other users start issuing requests to the `WorkInFrontEnd` POST method. At that point, response times increase dramatically (first arrow). Response times only improve after the volume of requests to the `WorkInFrontEnd` controller diminishes (second arrow).



Examine telemetry data and find correlations

The next image shows some of the metrics gathered to monitor resource utilization during the same interval. At first, few users are accessing the system. As more users connect, CPU utilization becomes very high (100%). Also notice that the network I/O rate initially goes up as CPU usage rises. But once CPU usage peaks, network I/O actually goes down. That's because the system can only handle a relatively small number of requests once the CPU is at capacity. As users disconnect, the CPU load tails off.

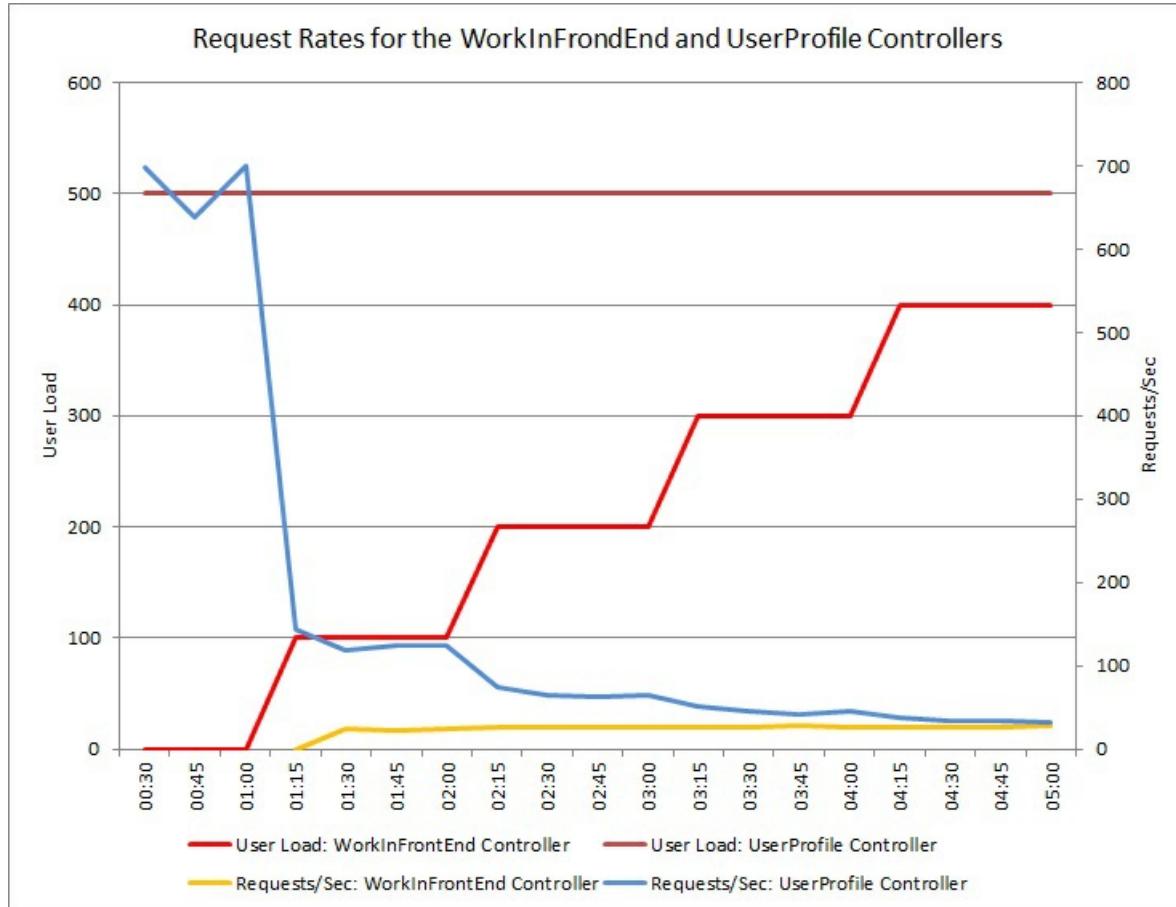


At this point, it appears the `Post` method in the `WorkInFrontEnd` controller is a prime candidate for closer examination. Further work in a controlled environment is needed to confirm the hypothesis.

Perform load testing

The next step is to perform tests in a controlled environment. For example, run a series of load tests that include and then omit each request in turn to see the effects.

The graph below shows the results of a load test performed against an identical deployment of the cloud service used in the previous tests. The test used a constant load of 500 users performing the `Get` operation in the `UserProfile` controller, along with a step load of users performing the `Post` operation in the `WorkInFrontEnd` controller.



Initially, the step load is 0, so the only active users are performing the `UserProfile` requests. The system is able to respond to approximately 500 requests per second. After 60 seconds, a load of 100 additional users starts sending POST requests to the `WorkInFrontEnd` controller. Almost immediately, the workload sent to the `UserProfile` controller drops to about 150 requests per second. This is due to the way the load-test runner functions. It waits for a response before sending the next request, so the longer it takes to receive a response, the lower the request rate.

As more users send POST requests to the `WorkInFrontEnd` controller, the response rate of the `UserProfile` controller continues to drop. But note that the volume of requests handled by the `WorkInFrontEnd` controller remains relatively constant. The saturation of the system becomes apparent as the overall rate of both requests tends toward a steady but low limit.

Review the source code

The final step is to look at the source code. The development team was aware that the `Post` method could take a considerable amount of time, which is why the original implementation used a separate thread. That solved the immediate problem, because the `Post` method did not block waiting for a long-running task to complete.

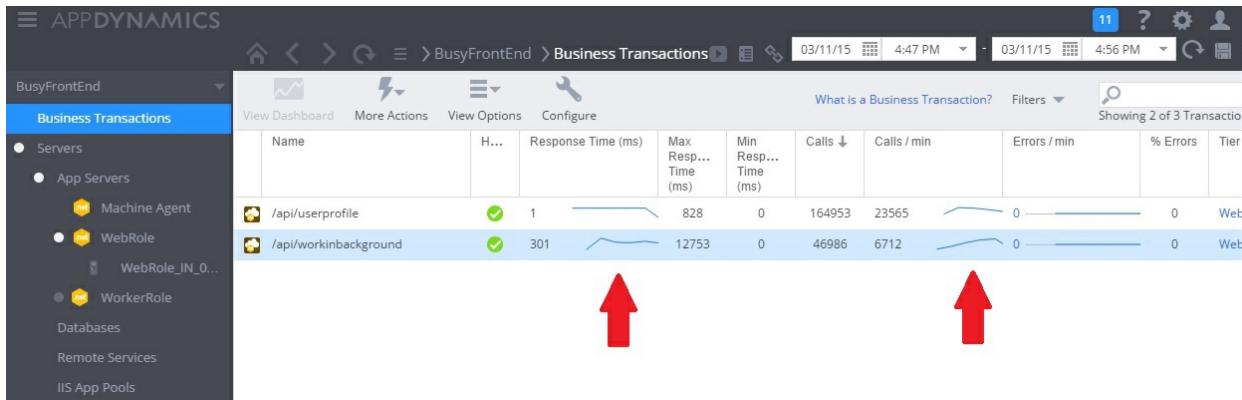
However, the work performed by this method still consumes CPU, memory, and other resources. Enabling this process to run asynchronously might actually damage performance, as users can trigger a large number of these operations simultaneously, in an uncontrolled manner. There is a limit to the number of threads that a server can run. Past this limit, the application is likely to get an exception when it tries to start a new thread.

NOTE

This doesn't mean you should avoid asynchronous operations. Performing an asynchronous await on a network call is a recommended practice. (See the [Synchronous I/O antipattern](#).) The problem here is that CPU-intensive work was spawned on another thread.

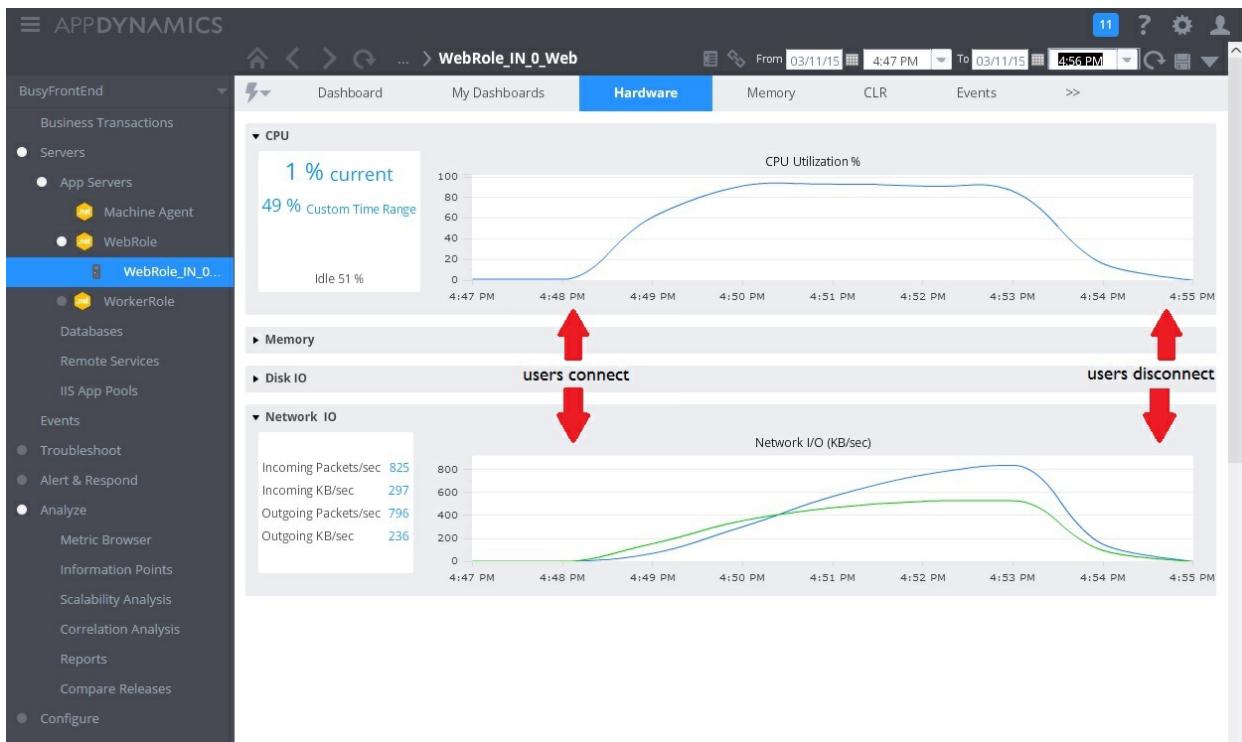
Implement the solution and verify the result

The following image shows performance monitoring after the solution was implemented. The load was similar to that shown earlier, but the response times for the `UserProfile` controller are now much faster. The volume of requests increased over the same duration, from 2,759 to 23,565.

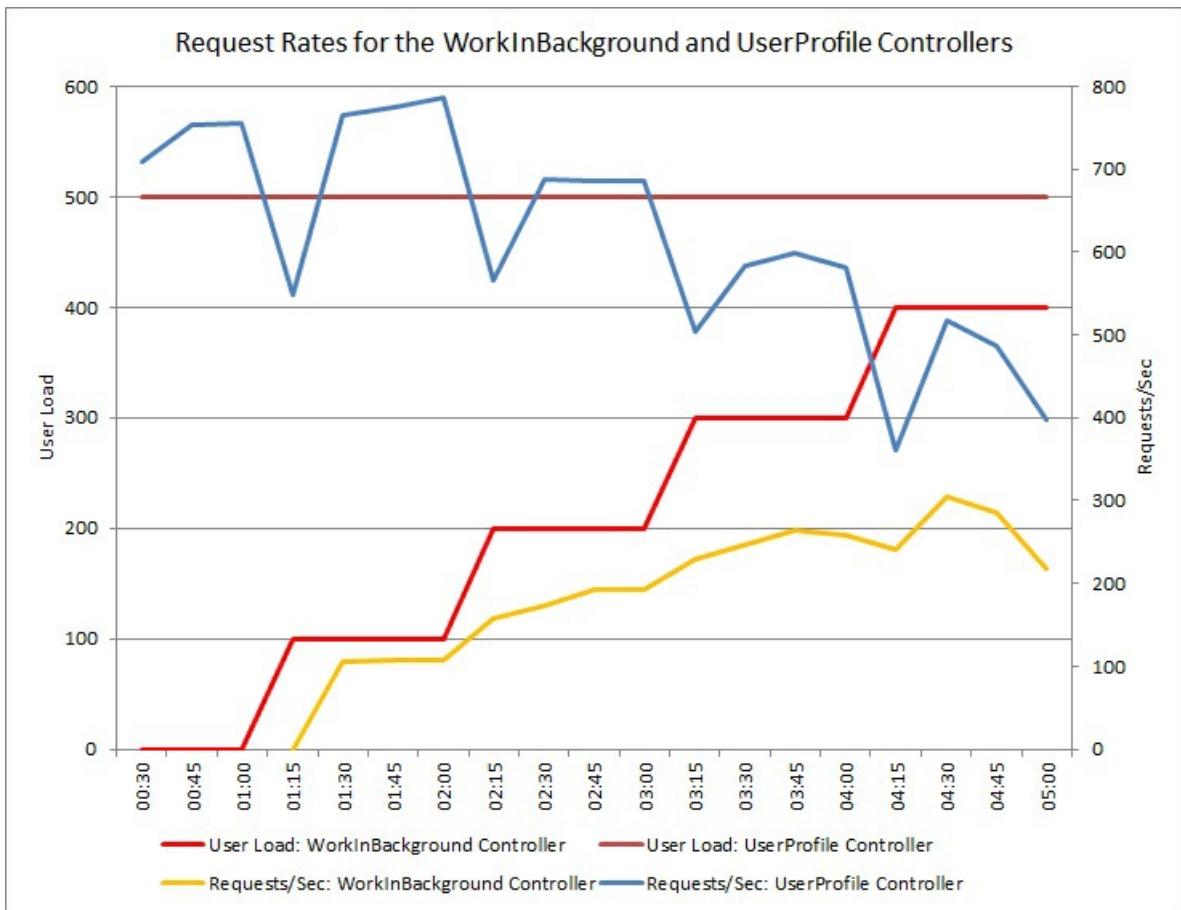


Note that the `WorkInBackground` controller also handled a much larger volume of requests. However, you can't make a direct comparison in this case, because the work being performed in this controller is very different from the original code. The new version simply queues a request, rather than performing a time consuming calculation. The main point is that this method no longer drags down the entire system under load.

CPU and network utilization also show the improved performance. The CPU utilization never reached 100%, and the volume of handled network requests was far greater than earlier, and did not tail off until the workload dropped.



The following graph shows the results of a load test. The overall volume of requests serviced is greatly improved compared to the earlier tests.



Related guidance

- [Autoscaling best practices](#)
- [Background jobs best practices](#)
- [Queue-Based Load Leveling pattern](#)
- [Web Queue Worker architecture style](#)

Chatty I/O antipattern

3/10/2022 • 9 minutes to read • [Edit Online](#)

The cumulative effect of a large number of I/O requests can have a significant impact on performance and responsiveness.

Problem description

Network calls and other I/O operations are inherently slow compared to compute tasks. Each I/O request typically has significant overhead, and the cumulative effect of numerous I/O operations can slow down the system. Here are some common causes of chatty I/O.

Reading and writing individual records to a database as distinct requests

The following example reads from a database of products. There are three tables, `Product`, `ProductSubcategory`, and `ProductPriceListHistory`. The code retrieves all of the products in a subcategory, along with the pricing information, by executing a series of queries:

1. Query the subcategory from the `ProductSubcategory` table.
2. Find all products in that subcategory by querying the `Product` table.
3. For each product, query the pricing data from the `ProductPriceListHistory` table.

The application uses [Entity Framework](#) to query the database. You can find the complete sample [here](#).

```
public async Task<IHttpActionResult> GetProductsInSubCategoryAsync(int subcategoryId)
{
    using (var context = GetContext())
    {
        // Get product subcategory.
        var productSubcategory = await context.ProductSubcategories
            .Where(psc => psc.ProductSubcategoryId == subcategoryId)
            .FirstOrDefaultAsync();

        // Find products in that category.
        productSubcategory.Product = await context.Products
            .Where(p => subcategoryId == p.ProductSubcategoryId)
            .ToListAsync();

        // Find price history for each product.
        foreach (var prod in productSubcategory.Product)
        {
            int productId = prod.ProductId;
            var productListPriceHistory = await context.ProductListPriceHistory
                .Where(pl => pl.ProductId == productId)
                .ToListAsync();
            prod.ProductListPriceHistory = productListPriceHistory;
        }
        return Ok(productSubcategory);
    }
}
```

This example shows the problem explicitly, but sometimes an O/RM can mask the problem, if it implicitly fetches child records one at a time. This is known as the "N+1 problem".

Implementing a single logical operation as a series of HTTP requests

This often happens when developers try to follow an object-oriented paradigm, and treat remote objects as if

they were local objects in memory. This can result in too many network round trips. For example, the following web API exposes the individual properties of `User` objects through individual HTTP GET methods.

```
public class UserController : ApiController
{
    [HttpGet]
    [Route("users/{id:int}/username")]
    public HttpResponseMessage GetUserName(int id)
    {
        ...
    }

    [HttpGet]
    [Route("users/{id:int}/gender")]
    public HttpResponseMessage GetGender(int id)
    {
        ...
    }

    [HttpGet]
    [Route("users/{id:int}/dateofbirth")]
    public HttpResponseMessage GetDateOfBirth(int id)
    {
        ...
    }
}
```

While there's nothing technically wrong with this approach, most clients will probably need to get several properties for each `User`, resulting in client code like the following.

```
HttpResponseMessage response = await client.GetAsync("users/1/username");
response.EnsureSuccessStatusCode();
var userName = await response.Content.ReadAsStringAsync();

response = await client.GetAsync("users/1/gender");
response.EnsureSuccessStatusCode();
var gender = await response.Content.ReadAsStringAsync();

response = await client.GetAsync("users/1/dateofbirth");
response.EnsureSuccessStatusCode();
var dob = await response.Content.ReadAsStringAsync();
```

Reading and writing to a file on disk

File I/O involves opening a file and moving to the appropriate point before reading or writing data. When the operation is complete, the file might be closed to save operating system resources. An application that continually reads and writes small amounts of information to a file will generate significant I/O overhead. Small write requests can also lead to file fragmentation, slowing subsequent I/O operations still further.

The following example uses a `FileStream` to write a `Customer` object to a file. Creating the `FileStream` opens the file, and disposing it closes the file. (The `using` statement automatically disposes the `FileStream` object.) If the application calls this method repeatedly as new customers are added, the I/O overhead can accumulate quickly.

```

private async Task SaveCustomerToFileAsync(Customer customer)
{
    using (Stream fileStream = new FileStream(CustomersFileName, FileMode.Append))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        byte [] data = null;
        using (MemoryStream memStream = new MemoryStream())
        {
            formatter.Serialize(memStream, customer);
            data = memStream.ToArray();
        }
        await fileStream.WriteAsync(data, 0, data.Length);
    }
}

```

How to fix the problem

Reduce the number of I/O requests by packaging the data into larger, fewer requests.

Fetch data from a database as a single query, instead of several smaller queries. Here's a revised version of the code that retrieves product information.

```

public async Task<IHttpActionResult> GetProductCategoryDetailsAsync(int subCategoryId)
{
    using (var context = GetContext())
    {
        var subCategory = await context.ProductSubcategories
            .Where(psc => psc.ProductSubcategoryId == subCategoryId)
            .Include("Product.ProductListPriceHistory")
            .FirstOrDefaultAsync();

        if (subCategory == null)
            return NotFound();

        return Ok(subCategory);
    }
}

```

Follow REST design principles for web APIs. Here's a revised version of the web API from the earlier example. Instead of separate GET methods for each property, there is a single GET method that returns the `User`. This results in a larger response body per request, but each client is likely to make fewer API calls.

```

public class UserController : ApiController
{
    [HttpGet]
    [Route("users/{id:int}")]
    public HttpResponseMessage GetUser(int id)
    {
        ...
    }

    // Client code
    HttpResponseMessage response = await client.GetAsync("users/1");
    response.EnsureSuccessStatusCode();
    var user = await response.Content.ReadAsStringAsync();
}

```

For file I/O, consider buffering data in memory and then writing the buffered data to a file as a single operation. This approach reduces the overhead from repeatedly opening and closing the file, and helps to reduce fragmentation of the file on disk.

```

// Save a list of customer objects to a file
private async Task SaveCustomerListToFileAsync(List<Customer> customers)
{
    using (FileStream fileStream = new FileStream(CustomersFileName, FileMode.Append))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        foreach (var customer in customers)
        {
            byte[] data = null;
            using (MemoryStream memStream = new MemoryStream())
            {
                formatter.Serialize(memStream, customer);
                data = memStream.ToArray();
            }
            await fileStream.WriteAsync(data, 0, data.Length);
        }
    }
}

// In-memory buffer for customers.
List<Customer> customers = new List<Customers>();

// Create a new customer and add it to the buffer
var customer = new Customer(...);
customers.Add(customer);

// Add more customers to the list as they are created
...

// Save the contents of the list, writing all customers in a single operation
await SaveCustomerListToFileAsync(customers);

```

Considerations

- The first two examples make *fewer* I/O calls, but each one retrieves *more* information. You must consider the tradeoff between these two factors. The right answer will depend on the actual usage patterns. For example, in the web API example, it might turn out that clients often need just the user name. In that case, it might make sense to expose it as a separate API call. For more information, see the [Extraneous Fetching](#) antipattern.
- When reading data, do not make your I/O requests too large. An application should only retrieve the information that it is likely to use.
- Sometimes it helps to partition the information for an object into two chunks, *frequently accessed data* that accounts for most requests, and *less frequently accessed data* that is used rarely. Often the most frequently accessed data is a relatively small portion of the total data for an object, so returning just that portion can save significant I/O overhead.
- When writing data, avoid locking resources for longer than necessary, to reduce the chances of contention during a lengthy operation. If a write operation spans multiple data stores, files, or services, then adopt an eventually consistent approach. See [Data Consistency guidance](#).
- If you buffer data in memory before writing it, the data is vulnerable if the process crashes. If the data rate typically has bursts or is relatively sparse, it may be safer to buffer the data in an external durable queue such as [Event Hubs](#).
- Consider caching data that you retrieve from a service or a database. This can help to reduce the volume of I/O by avoiding repeated requests for the same data. For more information, see [Caching best practices](#).

How to detect the problem

Symptoms of chatty I/O include high latency and low throughput. End users are likely to report extended response times or failures caused by services timing out, due to increased contention for I/O resources.

You can perform the following steps to help identify the causes of any problems:

1. Perform process monitoring of the production system to identify operations with poor response times.
2. Perform load testing of each operation identified in the previous step.
3. During the load tests, gather telemetry data about the data access requests made by each operation.
4. Gather detailed statistics for each request sent to a data store.
5. Profile the application in the test environment to establish where possible I/O bottlenecks might be occurring.

Look for any of these symptoms:

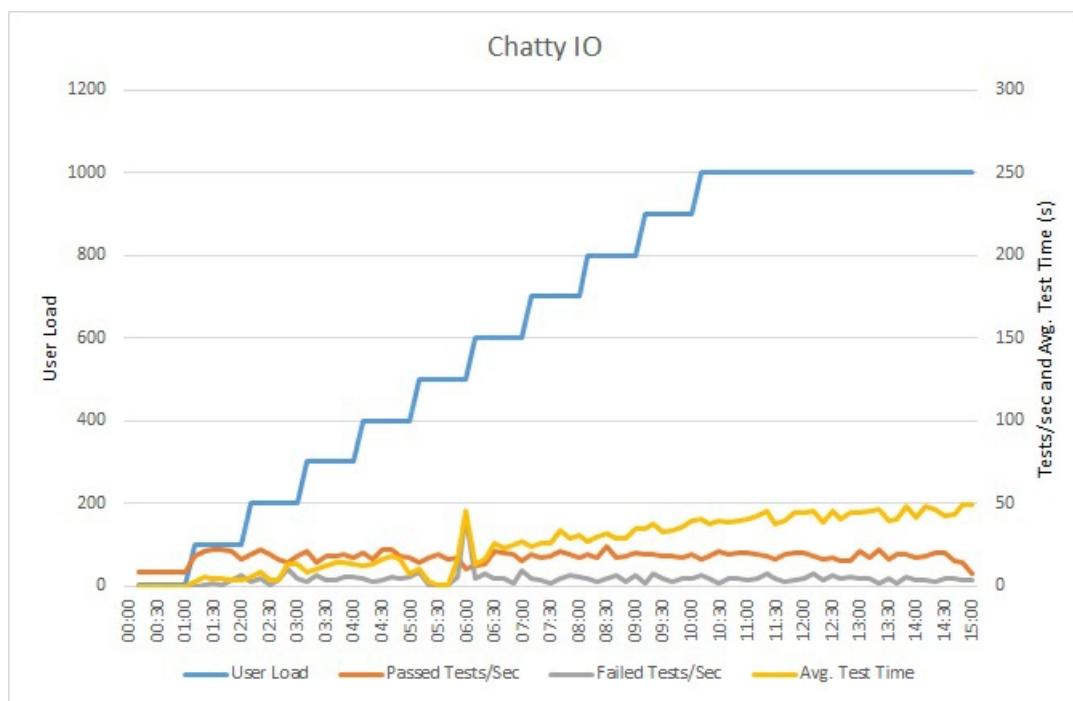
- A large number of small I/O requests made to the same file.
- A large number of small network requests made by an application instance to the same service.
- A large number of small requests made by an application instance to the same data store.
- Applications and services becoming I/O bound.

Example diagnosis

The following sections apply these steps to the example shown earlier that queries a database.

Load test the application

This graph shows the results of load testing. Median response time is measured in tens of seconds per request. The graph shows very high latency. With a load of 1000 users, a user might have to wait for nearly a minute to see the results of a query.



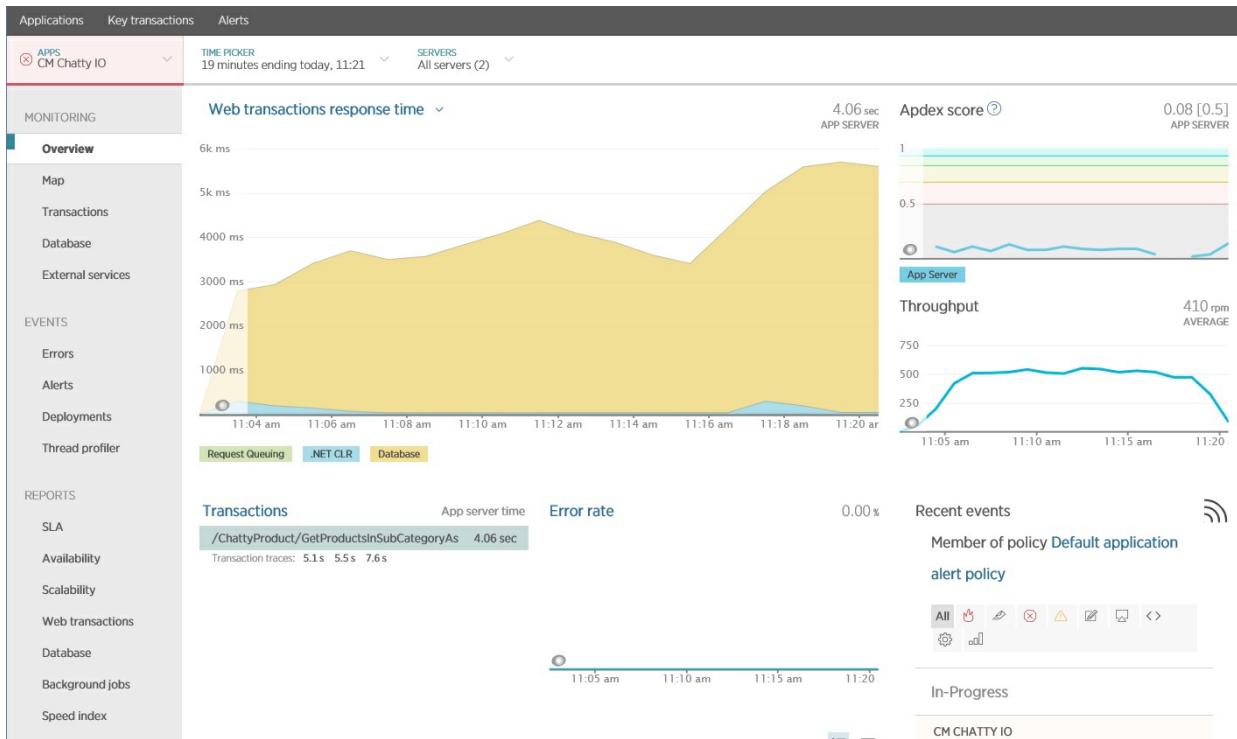
NOTE

The application was deployed as an Azure App Service web app, using Azure SQL Database. The load test used a simulated step workload of up to 1000 concurrent users. The database was configured with a connection pool supporting up to 1000 concurrent connections, to reduce the chance that contention for connections would affect the results.

Monitor the application

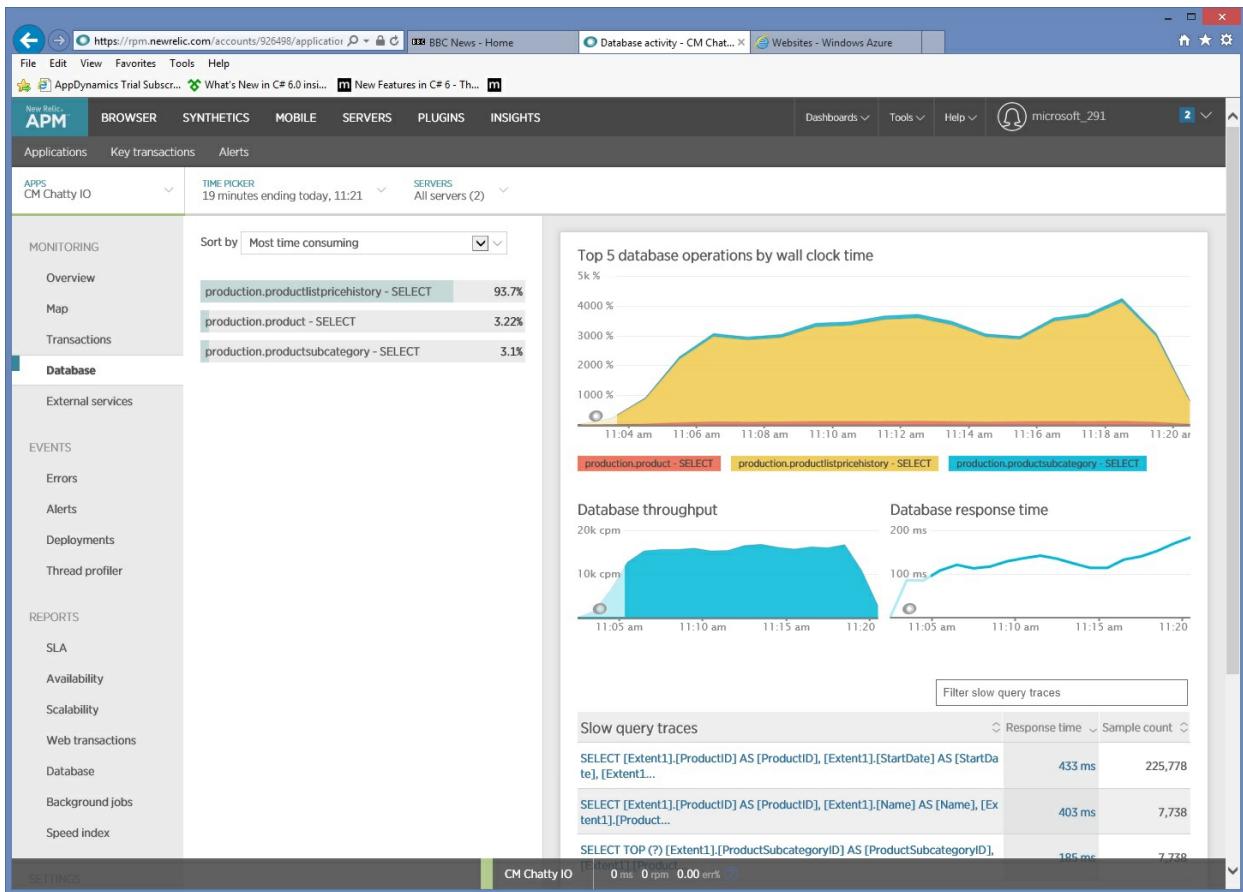
You can use an application performance monitoring (APM) package to capture and analyze the key metrics that might identify chatty I/O. Which metrics are important will depend on the I/O workload. For this example, the interesting I/O requests were the database queries.

The following image shows results generated using [New Relic APM](#). The average database response time peaked at approximately 5.6 seconds per request during the maximum workload. The system was able to support an average of 410 requests per minute throughout the test.

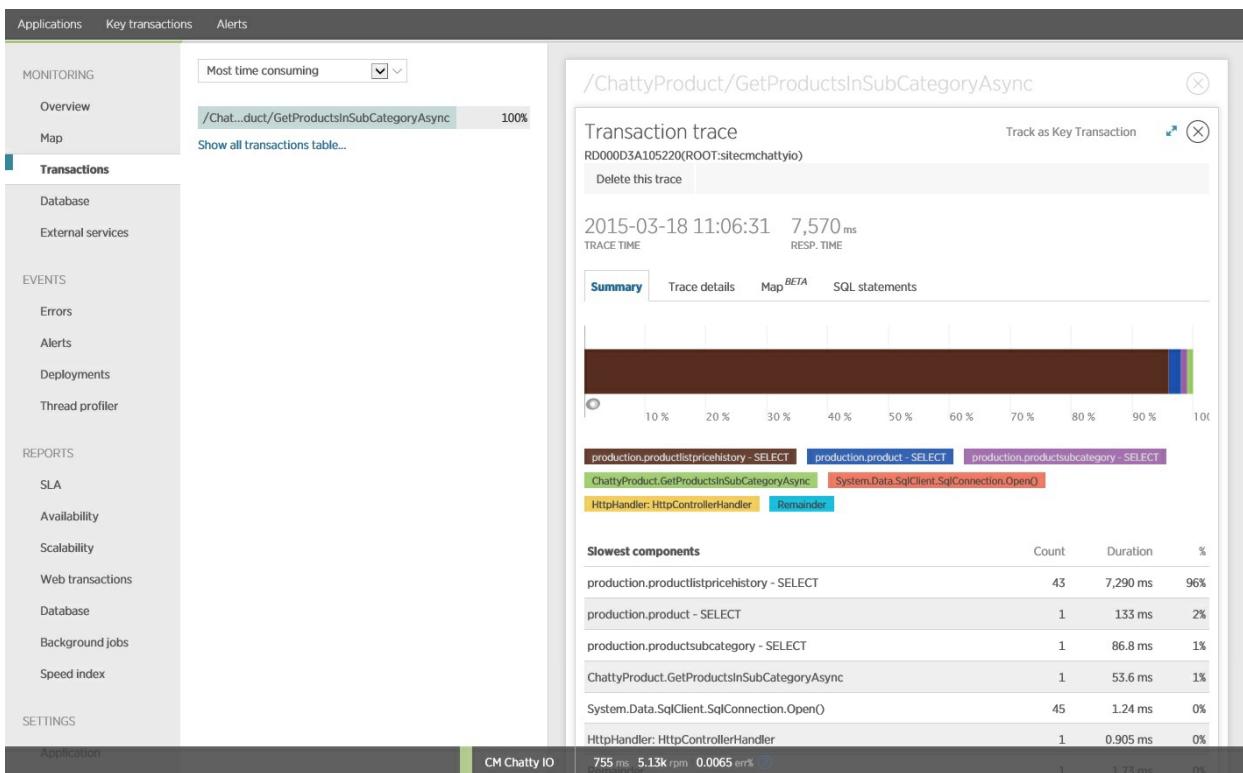


Gather detailed data access information

Digging deeper into the monitoring data shows the application executes three different SQL SELECT statements. These correspond to the requests generated by Entity Framework to fetch data from the `ProductListPriceHistory`, `Product`, and `ProductSubcategory` tables. Furthermore, the query that retrieves data from the `ProductListPriceHistory` table is by far the most frequently executed SELECT statement, by an order of magnitude.



It turns out that the `GetProductsInSubCategoryAsync` method, shown earlier, performs 45 SELECT queries. Each query causes the application to open a new SQL connection.



NOTE

This image shows trace information for the slowest instance of the `GetProductsInSubCategoryAsync` operation in the load test. In a production environment, it's useful to examine traces of the slowest instances, to see if there is a pattern that suggests a problem. If you just look at the average values, you might overlook problems that will get dramatically worse under load.

The next image shows the actual SQL statements that were issued. The query that fetches price information is run for each individual product in the product subcategory. Using a join would considerably reduce the number of database calls.

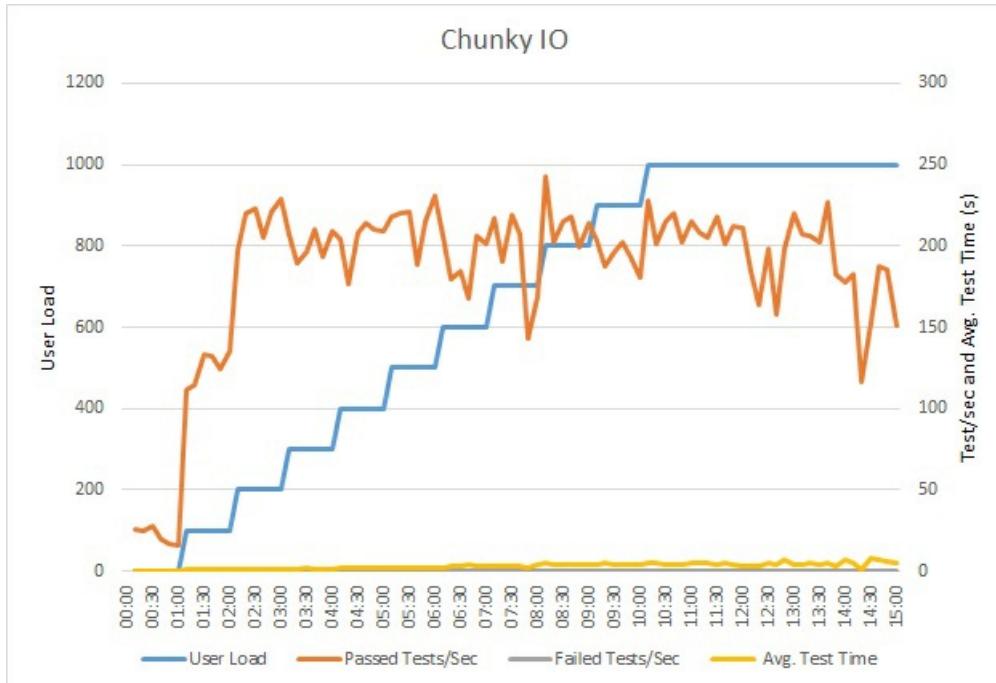
The screenshot shows the Application Insights interface. On the left, there's a sidebar with sections like MONITORING, Database, External services, EVENTS, Deployments, Thread profiler, REPORTS, SLA, Availability, Scalability, Web transactions, Database, Background jobs, Speed index, and SETTINGS. The main area is titled '/ChattyProduct/GetProductsInSubCategoryAsync' and shows a 'Transaction trace' for RD000D3A105220(ROOT:sitecmchattyio). It lists three SQL statements:

- SELECT [Extent1].[ProductID] AS [ProductID], [Extent1].[StartDate] AS [Start Date], [Extent1].[EndDate] AS [End Date], [Extent1].[ListPrice] AS [List Price] FROM [Production].[ProductListPriceHistory] AS [Extent1] WHERE [Extent1].[ProductID] = @p_1inq_0
- SELECT TOP (?) [Extent1].[ProductSubcategoryId] AS [ProductSubcategoryId], [Extent1].[ProductCategoryId] AS [ProductCategoryId], [Extent1].[Name] AS [Name] FROM [Production].[ProductSubcategory] AS [Extent1] WHERE [Extent1].[ProductSubcategoryId] = @p_1inq_0
- SELECT [Extent1].[ProductID] AS [ProductID], [Extent1].[Name] AS [Name], [Extent1].[ProductNumber] AS [ProductNumber], [Extent1].[ListPrice] AS [List Price], [Extent1].[ProductSubcategoryId] AS [ProductSubcategoryId] FROM [Production].[Product] AS [Extent1] WHERE @p_1inq_0 = [Extent1].[ProductSubcategoryId]

If you are using an O/RM, such as Entity Framework, tracing the SQL queries can provide insight into how the O/RM translates programmatic calls into SQL statements, and indicate areas where data access might be optimized.

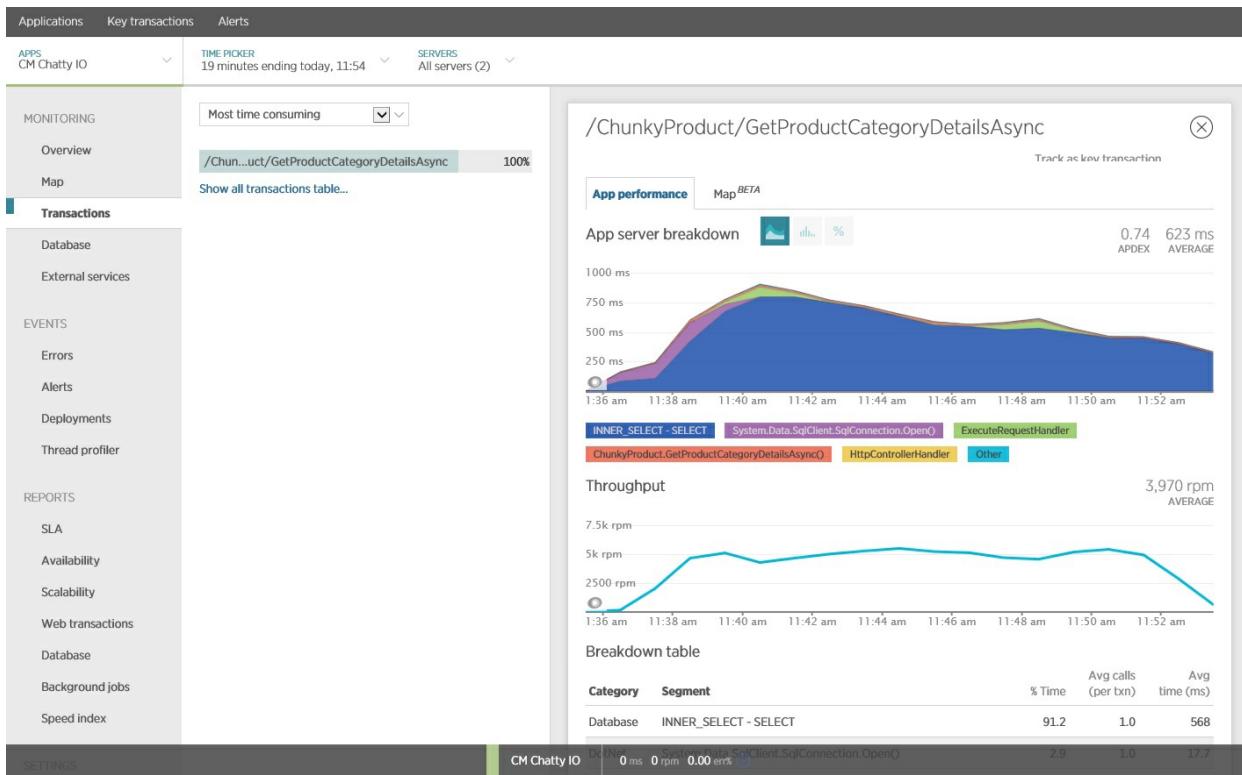
Implement the solution and verify the result

Rewriting the call to Entity Framework produced the following results.



This load test was performed on the same deployment, using the same load profile. This time the graph shows much lower latency. The average request time at 1000 users is between 5 and 6 seconds, down from nearly a minute.

This time the system supported an average of 3,970 requests per minute, compared to 410 for the earlier test.



Tracing the SQL statement shows that all the data is fetched in a single SELECT statement. Although this query is considerably more complex, it is performed only once per operation. And while complex joins can become expensive, relational database systems are optimized for this type of query.



Related resources

- [API Design best practices](#)
- [Caching best practices](#)
- [Data Consistency Primer](#)
- [Extraneous Fetching antipattern](#)
- [No Caching antipattern](#)

Extraneous Fetching antipattern

3/10/2022 • 10 minutes to read • [Edit Online](#)

Anti-patterns are common design flaws that can break your software or applications under stress situations and should not be overlooked. In an *extraneous fetching antipattern*, more than needed data is retrieved for a business operation, often resulting in unnecessary I/O overhead and reduced responsiveness.

Examples of extraneous fetching antipattern

This antipattern can occur if the application tries to minimize I/O requests by retrieving all of the data that it *might* need. This is often a result of overcompensating for the [Chatty I/O](#) antipattern. For example, an application might fetch the details for every product in a database. But the user may need just a subset of the details (some may not be relevant to customers), and probably doesn't need to see *all* of the products at once. Even if the user is browsing the entire catalog, it would make sense to paginate the results—showing 20 at a time, for example.

Another source of this problem is following poor programming or design practices. For example, the following code uses Entity Framework to fetch the complete details for every product. Then it filters the results to return only a subset of the fields, discarding the rest. You can find the complete sample [here](#).

```
public async Task<IHttpActionResult> GetAllFieldsAsync()
{
    using (var context = new AdventureWorksContext())
    {
        // Execute the query. This happens at the database.
        var products = await context.Products.ToListAsync();

        // Project fields from the query results. This happens in application memory.
        var result = products.Select(p => new ProductInfo { Id = p.ProductId, Name = p.Name });
        return Ok(result);
    }
}
```

In the next example, the application retrieves data to perform an aggregation that could be done by the database instead. The application calculates total sales by getting every record for all orders sold, and then computing the sum over those records. You can find the complete sample [here](#).

```
public async Task<IHttpActionResult> AggregateOnClientAsync()
{
    using (var context = new AdventureWorksContext())
    {
        // Fetch all order totals from the database.
        var orderAmounts = await context.SalesOrderHeaders.Select(soh => soh.TotalDue).ToListAsync();

        // Sum the order totals in memory.
        var total = orderAmounts.Sum();
        return Ok(total);
    }
}
```

The next example shows a subtle problem caused by the way Entity Framework uses LINQ to Entities.

```

var query = from p in context.Products.AsEnumerable()
            where p.SellStartDate < DateTime.Now.AddDays(-7) // AddDays cannot be mapped by LINQ to Entities
            select ...;

List<Product> products = query.ToList();

```

The application is trying to find products with a `SellStartDate` more than a week old. In most cases, LINQ to Entities would translate a `where` clause to a SQL statement that is executed by the database. In this case, however, LINQ to Entities cannot map the `AddDays` method to SQL. Instead, every row from the `Product` table is returned, and the results are filtered in memory.

The call to `AsEnumerable` is a hint that there is a problem. This method converts the results to an `IEnumerable` interface. Although `IEnumerable` supports filtering, the filtering is done on the *client* side, not the database. By default, LINQ to Entities uses `IQueryable`, which passes the responsibility for filtering to the data source.

How to fix extraneous fetching antipattern

Avoid fetching large volumes of data that may quickly become outdated or might be discarded, and only fetch the data needed for the operation being performed.

Instead of getting every column from a table and then filtering them, select the columns that you need from the database.

```

public async Task<IHttpActionResult> GetRequiredFieldsAsync()
{
    using (var context = new AdventureWorksContext())
    {
        // Project fields as part of the query itself
        var result = await context.Products
            .Select(p => new ProductInfo {Id = p.ProductId, Name = p.Name})
            .ToListAsync();
        return Ok(result);
    }
}

```

Similarly, perform aggregation in the database and not in application memory.

```

public async Task<IHttpActionResult> AggregateOnDatabaseAsync()
{
    using (var context = new AdventureWorksContext())
    {
        // Sum the order totals as part of the database query.
        var total = await context.SalesOrderHeaders.SumAsync(soh => soh.TotalDue);
        return Ok(total);
    }
}

```

When using Entity Framework, ensure that LINQ queries are resolved using the `IQueryable` interface and not `IEnumerable`. You may need to adjust the query to use only functions that can be mapped to the data source. The earlier example can be refactored to remove the `AddDays` method from the query, allowing filtering to be done by the database.

```

DateTime dateSince = DateTime.Now.AddDays(-7); // AddDays has been factored out.
var query = from p in context.Products
            where p.SellStartDate < dateSince // This criterion can be passed to the database by LINQ to
Entities
            select ...;

List<Product> products = query.ToList();

```

Considerations

- In some cases, you can improve performance by partitioning data horizontally. If different operations access different attributes of the data, horizontal partitioning may reduce contention. Often, most operations are run against a small subset of the data, so spreading this load may improve performance. See [Data partitioning](#).
- For operations that have to support unbounded queries, implement pagination and only fetch a limited number of entities at a time. For example, if a customer is browsing a product catalog, you can show one page of results at a time.
- When possible, take advantage of features built into the data store. For example, SQL databases typically provide aggregate functions.
- If you're using a data store that doesn't support a particular function, such as aggregation, you could store the calculated result elsewhere, updating the value as records are added or updated, so the application doesn't have to recalculate the value each time it's needed.
- If you see that requests are retrieving a large number of fields, examine the source code to determine whether all of these fields are necessary. Sometimes these requests are the result of poorly designed `SELECT *` query.
- Similarly, requests that retrieve a large number of entities may be sign that the application is not filtering data correctly. Verify that all of these entities are needed. Use database-side filtering if possible, for example, by using `WHERE` clauses in SQL.
- Offloading processing to the database is not always the best option. Only use this strategy when the database is designed or optimized to do so. Most database systems are highly optimized for certain functions, but are not designed to act as general-purpose application engines. For more information, see the [Busy Database antipattern](#).

How to detect extraneous fetching antipattern

Symptoms of extraneous fetching include high latency and low throughput. If the data is retrieved from a data store, increased contention is also probable. End users are likely to report extended response times or failures caused by services timing out. These failures could return HTTP 500 (Internal Server) errors or HTTP 503 (Service Unavailable) errors. Examine the event logs for the web server, which likely contain more detailed information about the causes and circumstances of the errors.

The symptoms of this antipattern and some of the telemetry obtained might be very similar to those of the [Monolithic Persistence antipattern](#).

You can perform the following steps to help identify the cause:

1. Identify slow workloads or transactions by performing load-testing, process monitoring, or other methods of capturing instrumentation data.
2. Observe any behavioral patterns exhibited by the system. Are there particular limits in terms of transactions per second or volume of users?

3. Correlate the instances of slow workloads with behavioral patterns.
4. Identify the data stores being used. For each data source, run lower-level telemetry to observe the behavior of operations.
5. Identify any slow-running queries that reference these data sources.
6. Perform a resource-specific analysis of the slow-running queries and ascertain how the data is used and consumed.

Look for any of these symptoms:

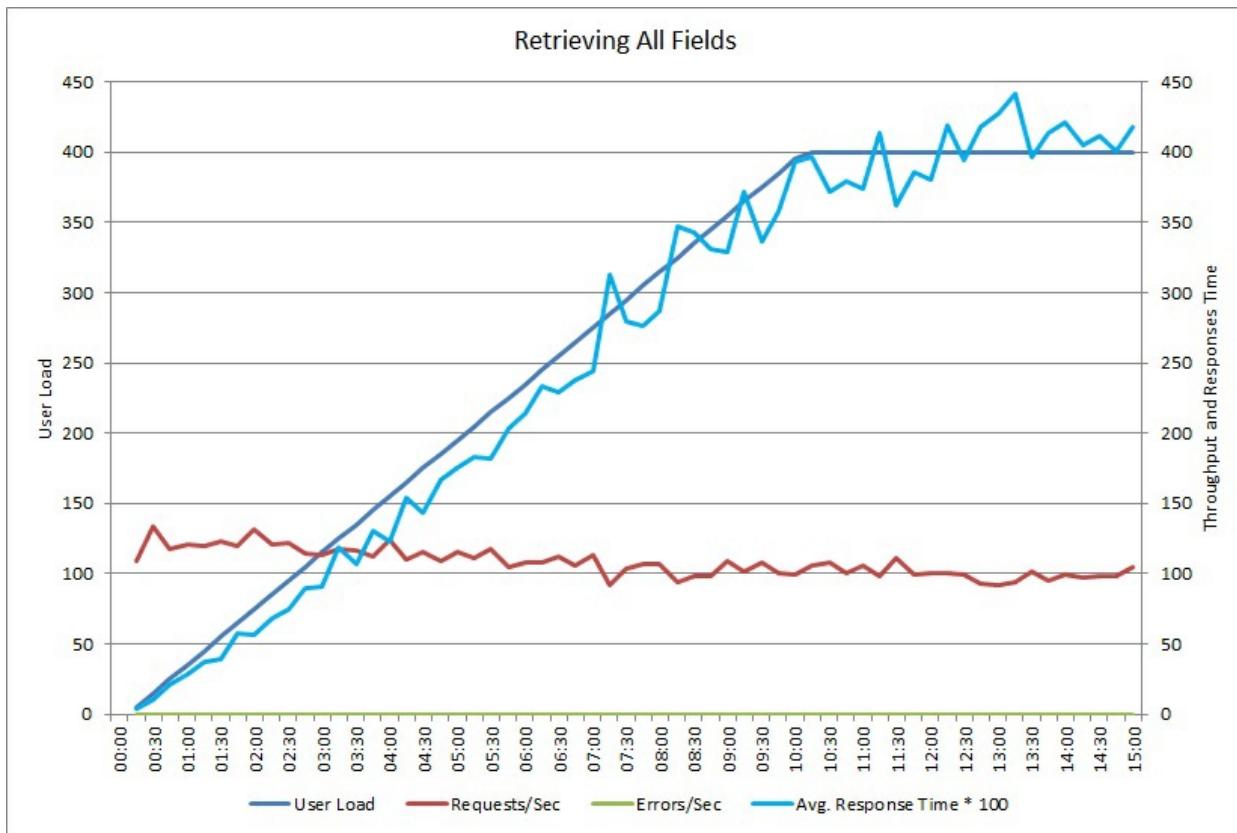
- Frequent, large I/O requests made to the same resource or data store.
- Contention in a shared resource or data store.
- An operation that frequently receives large volumes of data over the network.
- Applications and services spending significant time waiting for I/O to complete.

Example diagnosis

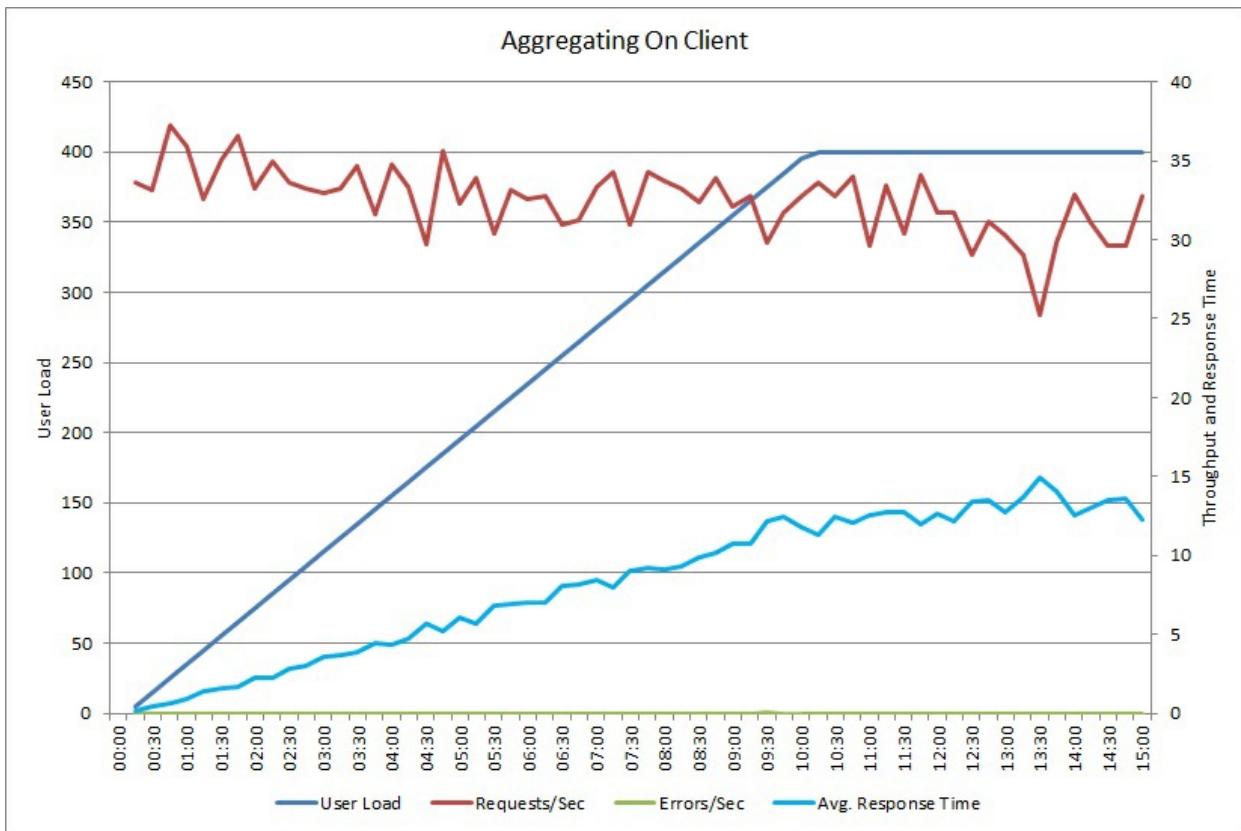
The following sections apply these steps to the previous examples.

Identify slow workloads

This graph shows performance results from a load test that simulated up to 400 concurrent users running the `GetAllFieldsAsync` method shown earlier. Throughput diminishes slowly as the load increases. Average response time goes up as the workload increases.



A load test for the `AggregateOnClientAsync` operation shows a similar pattern. The volume of requests is reasonably stable. The average response time increases with the workload, although more slowly than the previous graph.



Correlate slow workloads with behavioral patterns

Any correlation between regular periods of high usage and slowing performance can indicate areas of concern. Closely examine the performance profile of functionality that is suspected to be slow running, to determine whether it matches the load testing performed earlier.

Load test the same functionality using step-based user loads, to find the point where performance drops significantly or fails completely. If that point falls within the bounds of your expected real-world usage, examine how the functionality is implemented.

A slow operation is not necessarily a problem, if it is not being performed when the system is under stress, is not time critical, and does not negatively affect the performance of other important operations. For example, generating monthly operational statistics might be a long-running operation, but it can probably be performed as a batch process and run as a low-priority job. On the other hand, customers querying the product catalog is a critical business operation. Focus on the telemetry generated by these critical operations to see how the performance varies during periods of high usage.

Identify data sources in slow workloads

If you suspect that a service is performing poorly because of the way it retrieves data, investigate how the application interacts with the repositories it uses. Monitor the live system to see which sources are accessed during periods of poor performance.

For each data source, instrument the system to capture the following:

- The frequency that each data store is accessed.
- The volume of data entering and exiting the data store.
- The timing of these operations, especially the latency of requests.
- The nature and rate of any errors that occur while accessing each data store under typical load.

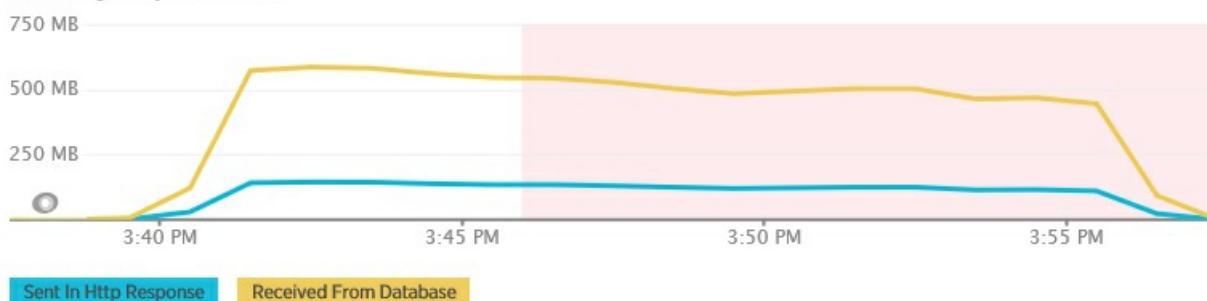
Compare this information against the volume of data being returned by the application to the client. Track the ratio of the volume of data returned by the data store against the volume of data returned to the client. If there is any large disparity, investigate to determine whether the application is fetching data that it doesn't need.

You may be able to capture this data by observing the live system and tracing the lifecycle of each user request,

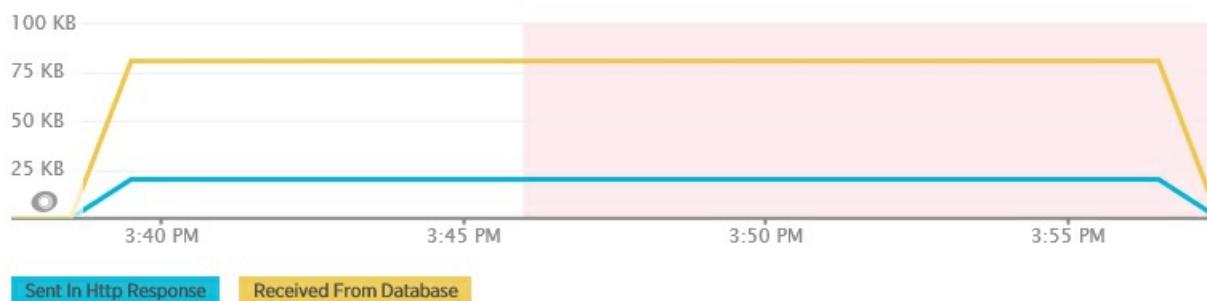
or you can model a series of synthetic workloads and run them against a test system.

The following graphs show telemetry captured using [New Relic APM](#) during a load test of the `GetAllFieldsAsync` method. Note the difference between the volumes of data received from the database and the corresponding HTTP responses.

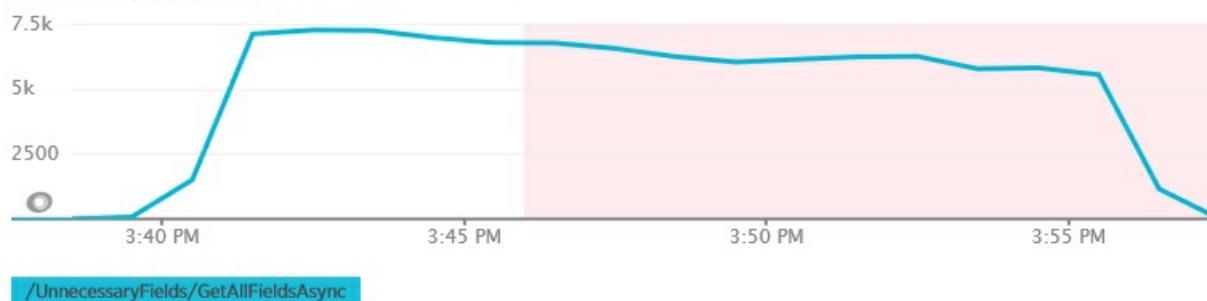
Total Bytes per Minute



Average Bytes per Transaction



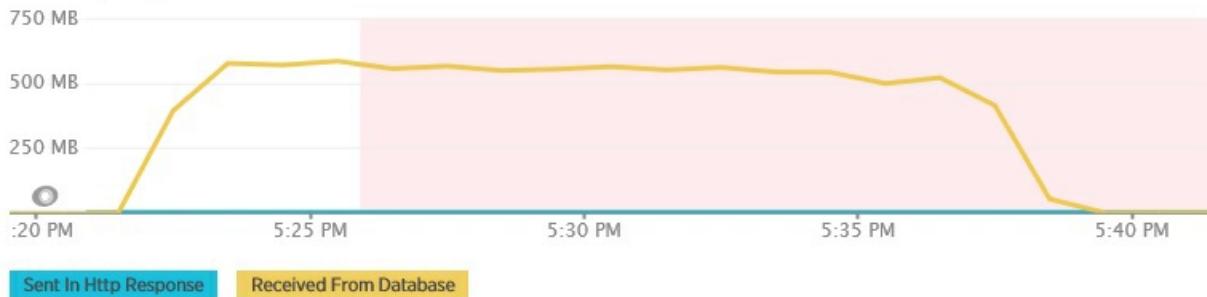
Requests per Minute



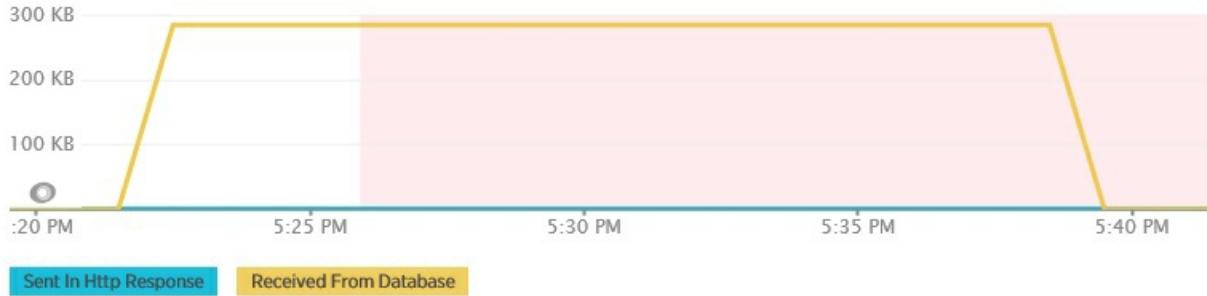
For each request, the database returned 80,503 bytes, but the response to the client only contained 19,855 bytes, about 25% of the size of the database response. The size of the data returned to the client can vary depending on the format. For this load test, the client requested JSON data. Separate testing using XML (not shown) had a response size of 35,655 bytes, or 44% of the size of the database response.

The load test for the `AggregateOnClientAsync` method shows more extreme results. In this case, each test performed a query that retrieved over 280 Kb of data from the database, but the JSON response was a mere 14 bytes. The wide disparity is because the method calculates an aggregated result from a large volume of data.

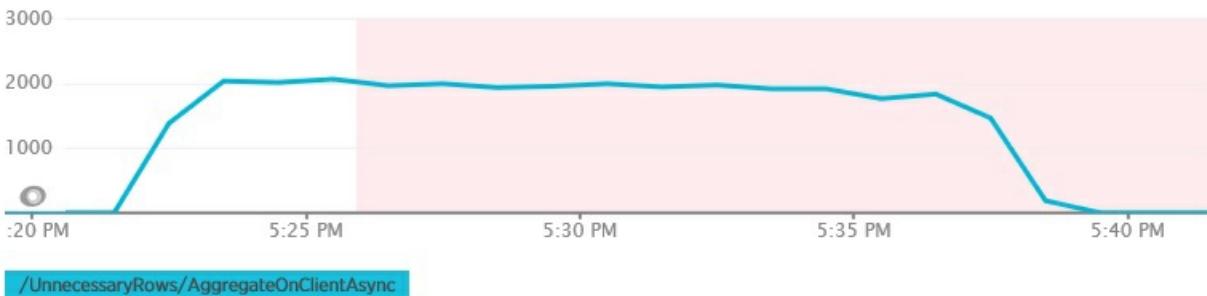
Total Bytes per Minute



Average Bytes per Transaction



Requests per Minute



Identify and analyze slow queries

Look for database queries that consume the most resources and take the most time to execute. You can add instrumentation to find the start and completion times for many database operations. Many data stores also provide in-depth information on how queries are performed and optimized. For example, the Query Performance pane in the Azure SQL Database management portal lets you select a query and view detailed runtime performance information. Here is the query generated by the `GetAllFieldsAsync` operation:

```

SELECT
    [Extent1].[ProductID] AS [ProductID],
    [Extent1].[Name] AS [Name],
    [Extent1].[ProductNumber] AS [ProductNumber],
    [Extent1].[MakeFlag] AS [MakeFlag],
    [Extent1].[FinishedGoodsFlag] AS [FinishedGoodsFlag],
    [Extent1].[Color] AS [Color],
    [Extent1].[SafetyStockLevel] AS [SafetyStockLevel],
    [Extent1].[ReorderPoint] AS [ReorderPoint],
    [Extent1].[StandardCost] AS [StandardCost],
    [Extent1].[ListPrice] AS [ListPrice],
    [Extent1].[Size] AS [Size],
    [Extent1].[SizeUnitMeasureCode] AS [SizeUnitMeasureCode],
    [Extent1].[WeightUnitMeasureCode] AS [WeightUnitMeasureCode],
    [Extent1].[Weight] AS [Weight],

```

Query Plan Details Query Plan

Resource Use

Resource	Total / sec	Total	Last Run	Minimum	Maximum
Duration (ms)	117	63450	40	0	3867
CPU (ms)	1	752	0	0	3
Logical Reads	25	13872	16	16	16
Physical Reads	0	0	0	0	0
Logical Writes	0	0	0	0	0

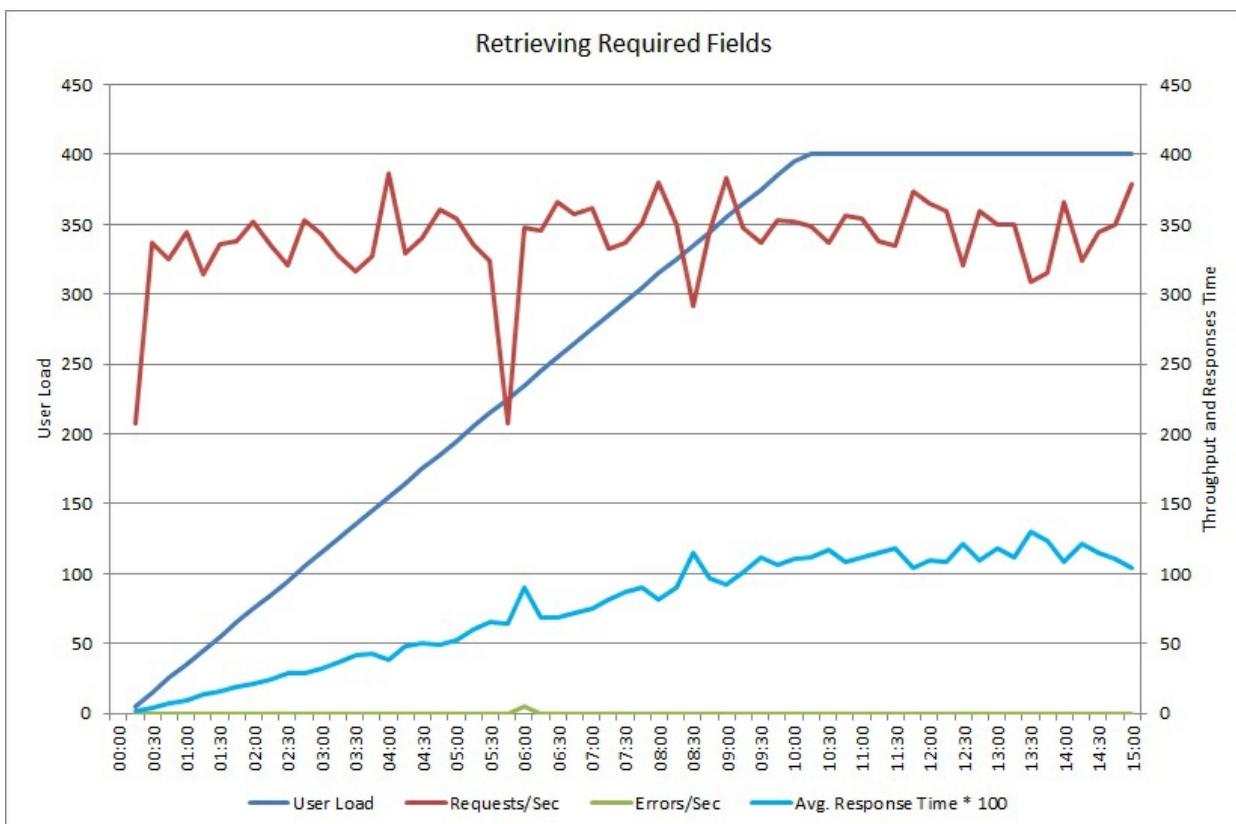
Plan Information

Advanced Information

Run Count	867	Plan Handle	0x0600E0004D00CD02A0EDD87D550000000100000000000000000000000000000000
Last Run Time	03/03/2015 15:32:25	SQL Handle	0x020000004D00CD02B85696A75DA7BEBF79E79259988F30C300000000
Plan Generation Count	1	Query Hash	0x05ACF4F9056ACADC
Time Plan Cached	03/03/2015 15:26:32	Query Plan Hash	0x0DA11AA10A268A7B

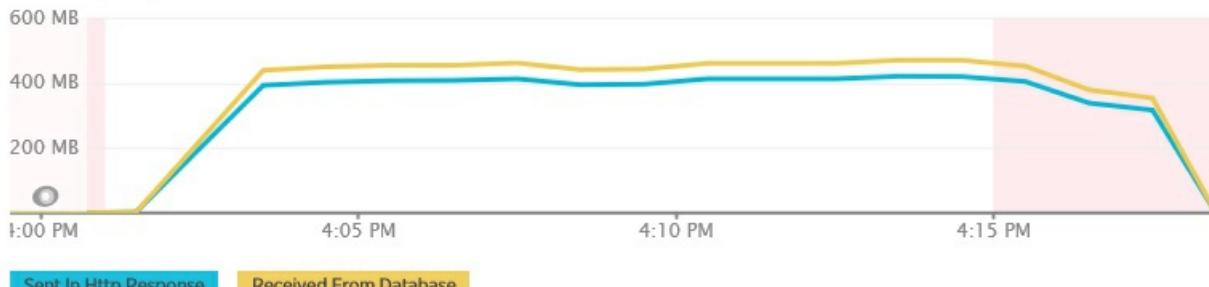
Implement the solution and verify the result

After changing the `GetRequiredFieldsAsync` method to use a SELECT statement on the database side, load testing showed the following results.

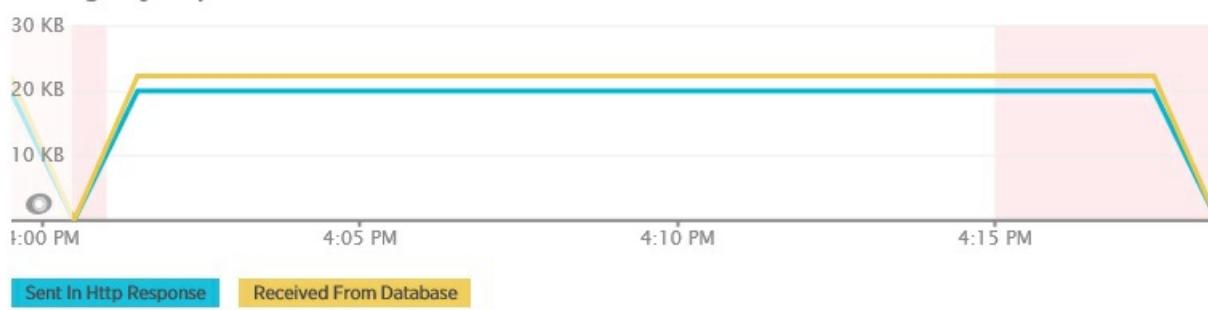


This load test used the same deployment and the same simulated workload of 400 concurrent users as before. The graph shows much lower latency. Response time rises with load to approximately 1.3 seconds, compared to 4 seconds in the previous case. The throughput is also higher at 350 requests per second compared to 100 earlier. The volume of data retrieved from the database now closely matches the size of the HTTP response messages.

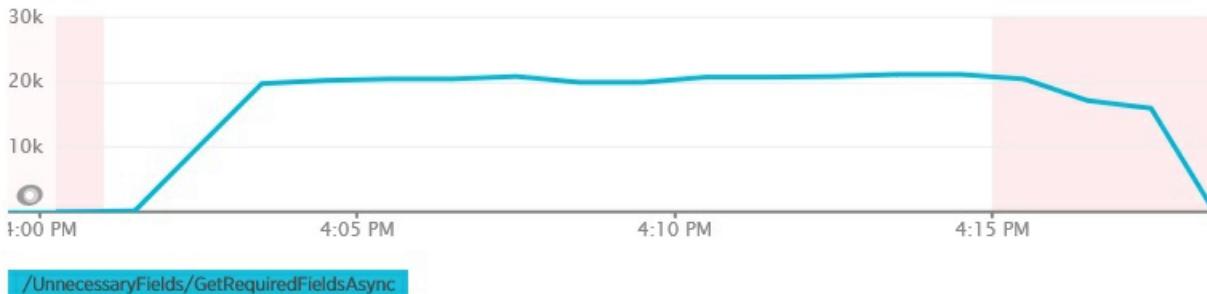
Total Bytes per Minute



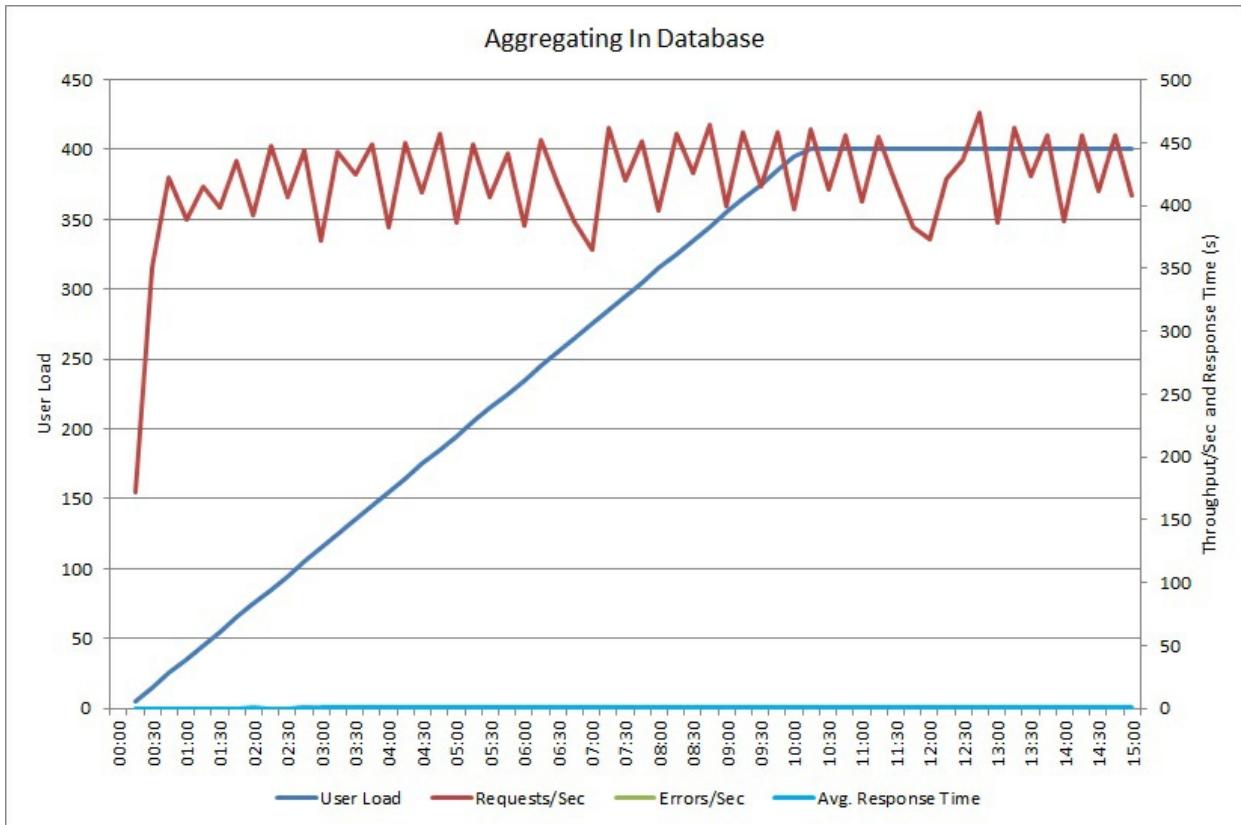
Average Bytes per Transaction



Requests per Minute



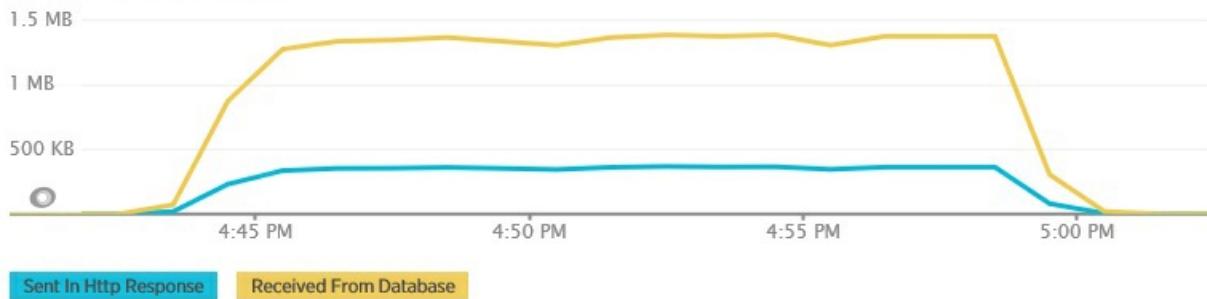
Load testing using the `AggregateOnDatabaseAsync` method generates the following results:



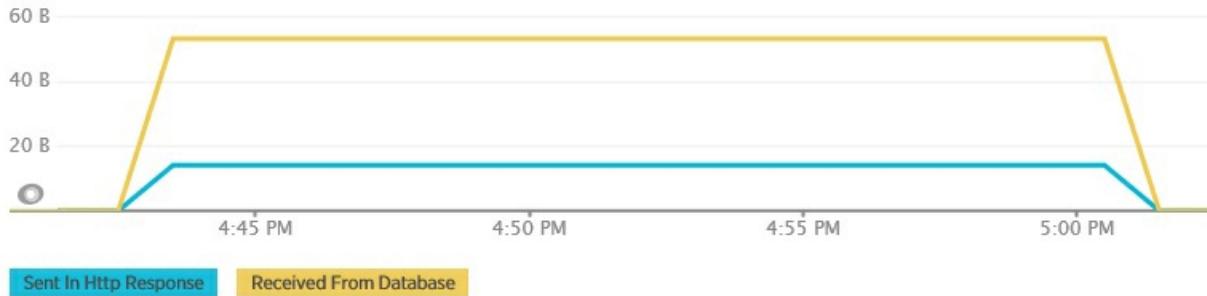
The average response time is now minimal. This is an order of magnitude improvement in performance, caused primarily by the large reduction in I/O from the database.

Here is the corresponding telemetry for the `AggregateOnDatabaseAsync` method. The amount of data retrieved from the database was vastly reduced, from over 280 Kb per transaction to 53 bytes. As a result, the maximum sustained number of requests per minute was raised from around 2,000 to over 25,000.

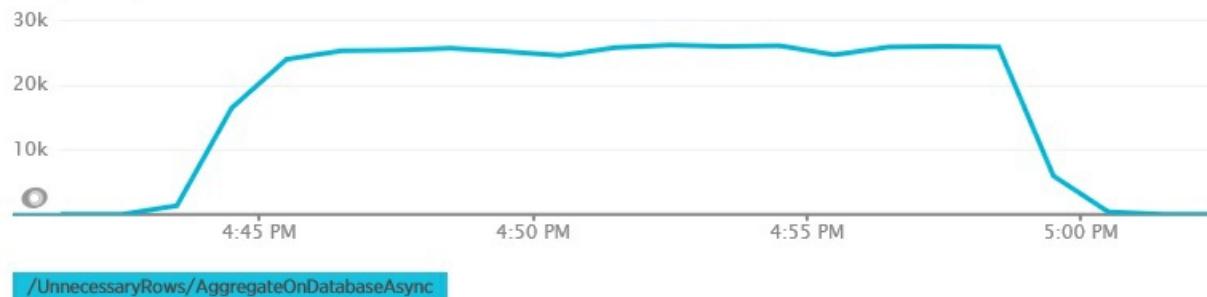
Total Bytes per Minute



Average Bytes per Transaction



Requests per Minute



Related resources

- [Busy Database antipattern](#)
- [Chatty I/O antipattern](#)
- [Data partitioning best practices](#)

Improper Instantiation antipattern

3/10/2022 • 7 minutes to read • [Edit Online](#)

Sometimes new instances of a class are continually created, when it is meant to be created once and then shared. This behavior can hurt performance, and is called an *improper instantiation antipattern*. An antipattern is a common response to a recurring problem that is usually ineffective and may even be counter-productive.

Problem description

Many libraries provide abstractions of external resources. Internally, these classes typically manage their own connections to the resource, acting as brokers that clients can use to access the resource. Here are some examples of broker classes that are relevant to Azure applications:

- `System.Net.Http.HttpClient`. Communicates with a web service using HTTP.
- `Microsoft.ServiceBus.Messaging.QueueClient`. Posts and receives messages to a Service Bus queue.
- `Microsoft.Azure.Documents.Client.DocumentClient`. Connects to a Cosmos DB instance.
- `StackExchange.Redis.ConnectionMultiplexer`. Connects to Redis, including Azure Cache for Redis.

These classes are intended to be instantiated once and reused throughout the lifetime of an application. However, it's a common misunderstanding that these classes should be acquired only as necessary and released quickly. (The ones listed here happen to be .NET libraries, but the pattern is not unique to .NET.) The following ASP.NET example creates an instance of `HttpClient` to communicate with a remote service. You can find the complete sample [here](#).

```
public class NewHttpClientInstancePerRequestController : ApiController
{
    // This method creates a new instance of HttpClient and disposes it for every call to GetProductAsync.
    public async Task<Product> GetProductAsync(string id)
    {
        using (var httpClient = new HttpClient())
        {
            var hostName = HttpContext.Current.Request.Url.Host;
            var result = await httpClient.GetStringAsync(string.Format("http://{0}:8080/api/...", hostName));
            return new Product { Name = result };
        }
    }
}
```

In a web application, this technique is not scalable. A new `HttpClient` object is created for each user request. Under heavy load, the web server may exhaust the number of available sockets, resulting in `SocketException` errors.

This problem is not restricted to the `HttpClient` class. Other classes that wrap resources or are expensive to create might cause similar issues. The following example creates an instance of the `ExpensiveToCreateService` class. Here the issue is not necessarily socket exhaustion, but simply how long it takes to create each instance. Continually creating and destroying instances of this class might adversely affect the scalability of the system.

```

public class NewServiceInstancePerRequestController : ApiController
{
    public async Task<Product> GetProductAsync(string id)
    {
        var expensiveToCreateService = new ExpensiveToCreateService();
        return await expensiveToCreateService.GetProductByIdAsync(id);
    }
}

public class ExpensiveToCreateService
{
    public ExpensiveToCreateService()
    {
        // Simulate delay due to setup and configuration of ExpensiveToCreateService
        Thread.Sleep(Int32.MaxValue / 100);
    }
    ...
}

```

How to fix improper instantiation antipattern

If the class that wraps the external resource is shareable and thread-safe, create a shared singleton instance or a pool of reusable instances of the class.

The following example uses a static `HttpClient` instance, thus sharing the connection across all requests.

```

public class SingleHttpClientInstanceController : ApiController
{
    private static readonly HttpClient httpClient;

    static SingleHttpClientInstanceController()
    {
        httpClient = new HttpClient();
    }

    // This method uses the shared instance of HttpClient for every call to GetProductAsync.
    public async Task<Product> GetProductAsync(string id)
    {
        var hostName = HttpContext.Current.Request.Url.Host;
        var result = await httpClient.GetStringAsync(string.Format("http://{0}:8080/api/...", hostName));
        return new Product { Name = result };
    }
}

```

Considerations

- The key element of this antipattern is repeatedly creating and destroying instances of a *shareable* object. If a class is not shareable (not thread-safe), then this antipattern does not apply.
- The type of shared resource might dictate whether you should use a singleton or create a pool. The `HttpClient` class is designed to be shared rather than pooled. Other objects might support pooling, enabling the system to spread the workload across multiple instances.
- Objects that you share across multiple requests *must* be thread-safe. The `HttpClient` class is designed to be used in this manner, but other classes might not support concurrent requests, so check the available documentation.
- Be careful about setting properties on shared objects, as this can lead to race conditions. For example, setting `DefaultRequestHeaders` on the `HttpClient` class before each request can create a race condition. Set such properties once (for example, during startup), and create separate instances if you need to

configure different settings.

- Some resource types are scarce and should not be held onto. Database connections are an example. Holding an open database connection that is not required may prevent other concurrent users from gaining access to the database.
- In the .NET Framework, many objects that establish connections to external resources are created by using static factory methods of other classes that manage these connections. These objects are intended to be saved and reused, rather than disposed and re-created. For example, in Azure Service Bus, the `QueueClient` object is created through a `MessagingFactory` object. Internally, the `MessagingFactory` manages connections. For more information, see [Best Practices for performance improvements using Service Bus Messaging](#).

How to detect improper instantiation antipattern

Symptoms of this problem include a drop in throughput or an increased error rate, along with one or more of the following:

- An increase in exceptions that indicate exhaustion of resources such as sockets, database connections, file handles, and so on.
- Increased memory use and garbage collection.
- An increase in network, disk, or database activity.

You can perform the following steps to help identify this problem:

1. Performing process monitoring of the production system, to identify points when response times slow down or the system fails due to lack of resources.
2. Examine the telemetry data captured at these points to determine which operations might be creating and destroying resource-consuming objects.
3. Load test each suspected operation, in a controlled test environment rather than the production system.
4. Review the source code and examine the how broker objects are managed.

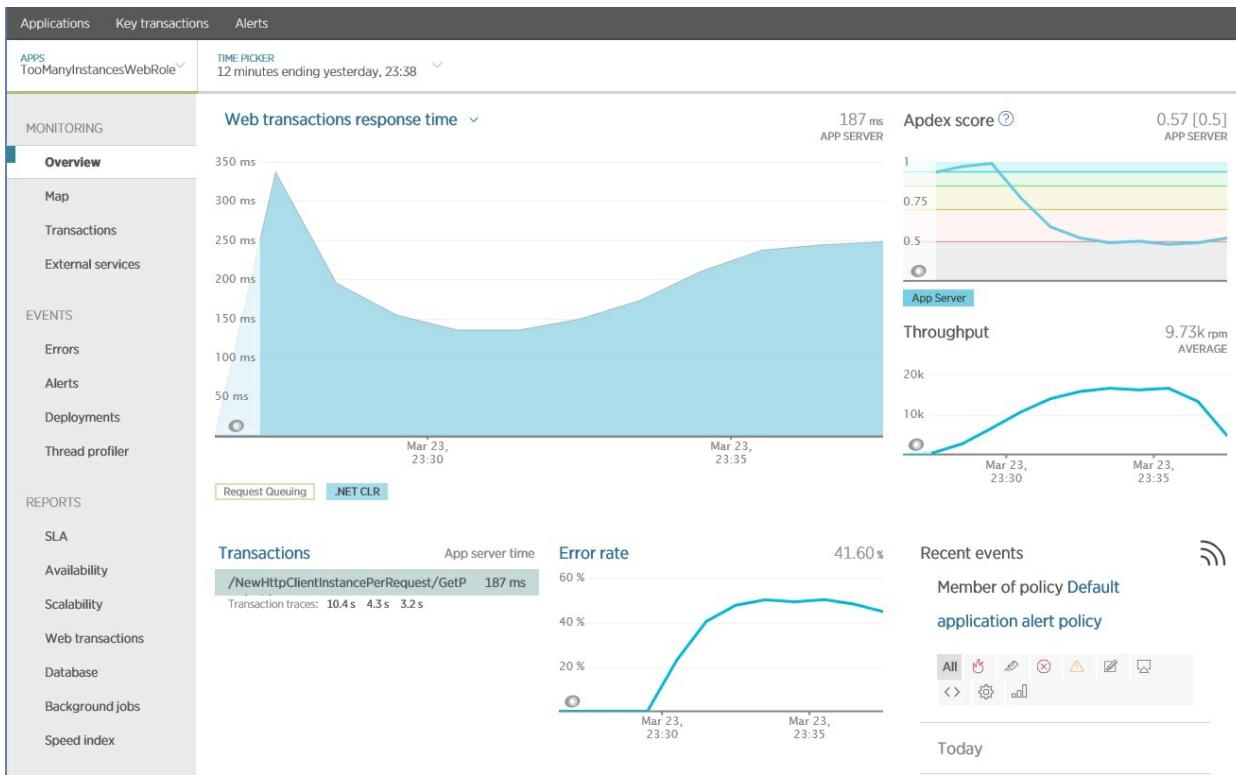
Look at stack traces for operations that are slow-running or that generate exceptions when the system is under load. This information can help to identify how these operations are using resources. Exceptions can help to determine whether errors are caused by shared resources being exhausted.

Example diagnosis

The following sections apply these steps to the sample application described earlier.

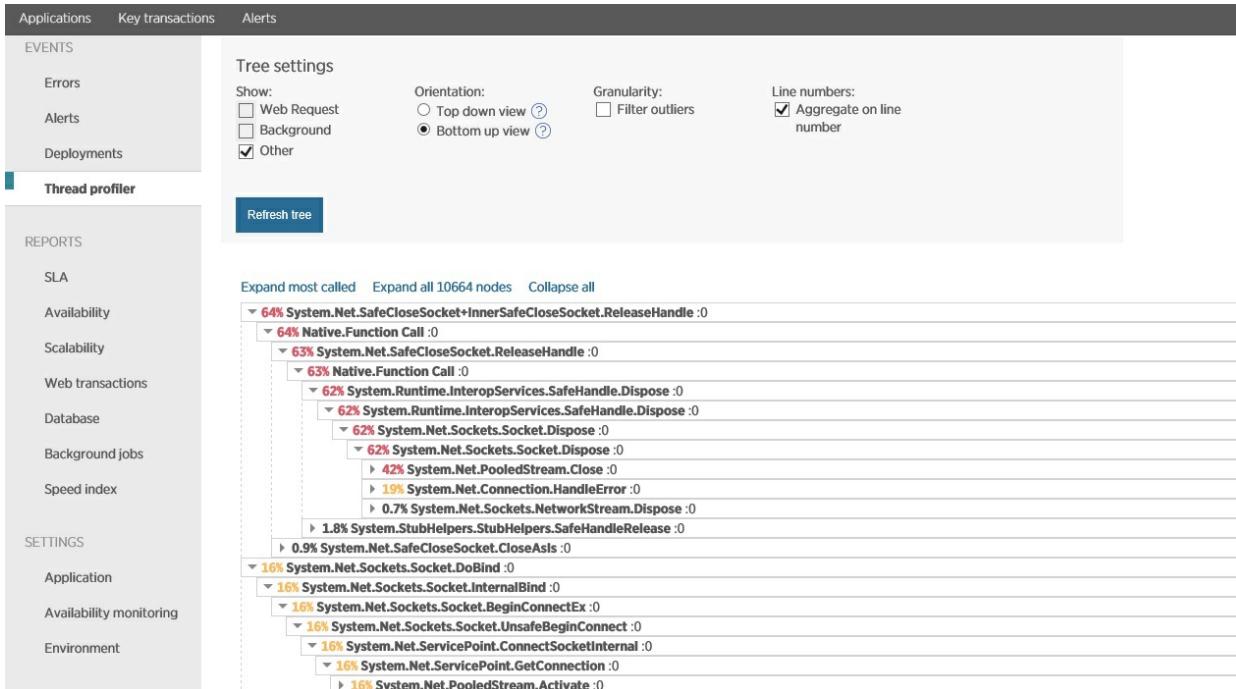
Identify points of slowdown or failure

The following image shows results generated using [New Relic APM](#), showing operations that have a poor response time. In this case, the `GetProductAsync` method in the `NewHttpClientInstancePerRequest` controller is worth investigating further. Notice that the error rate also increases when these operations are running.



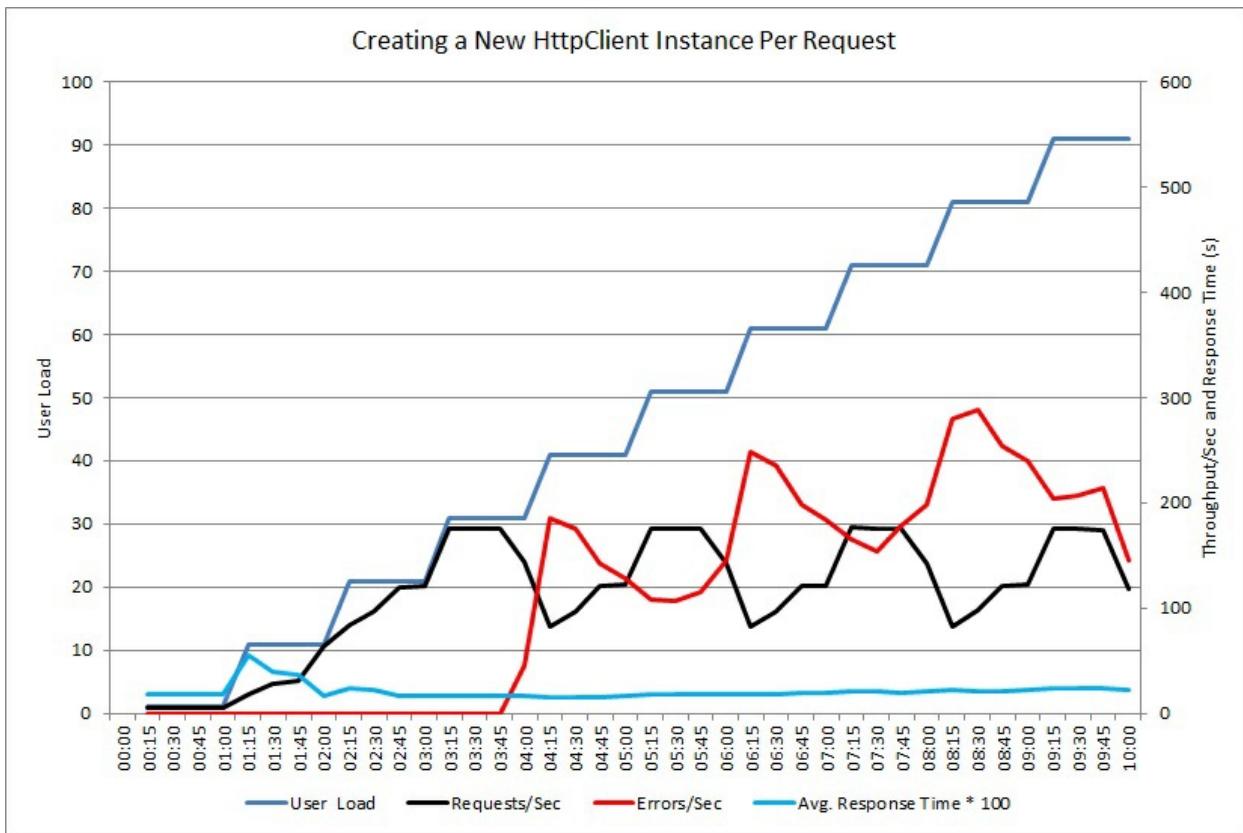
Examine telemetry data and find correlations

The next image shows data captured using thread profiling, over the same period corresponding as the previous image. The system spends a significant time opening socket connections, and even more time closing them and handling socket exceptions.



Performing load testing

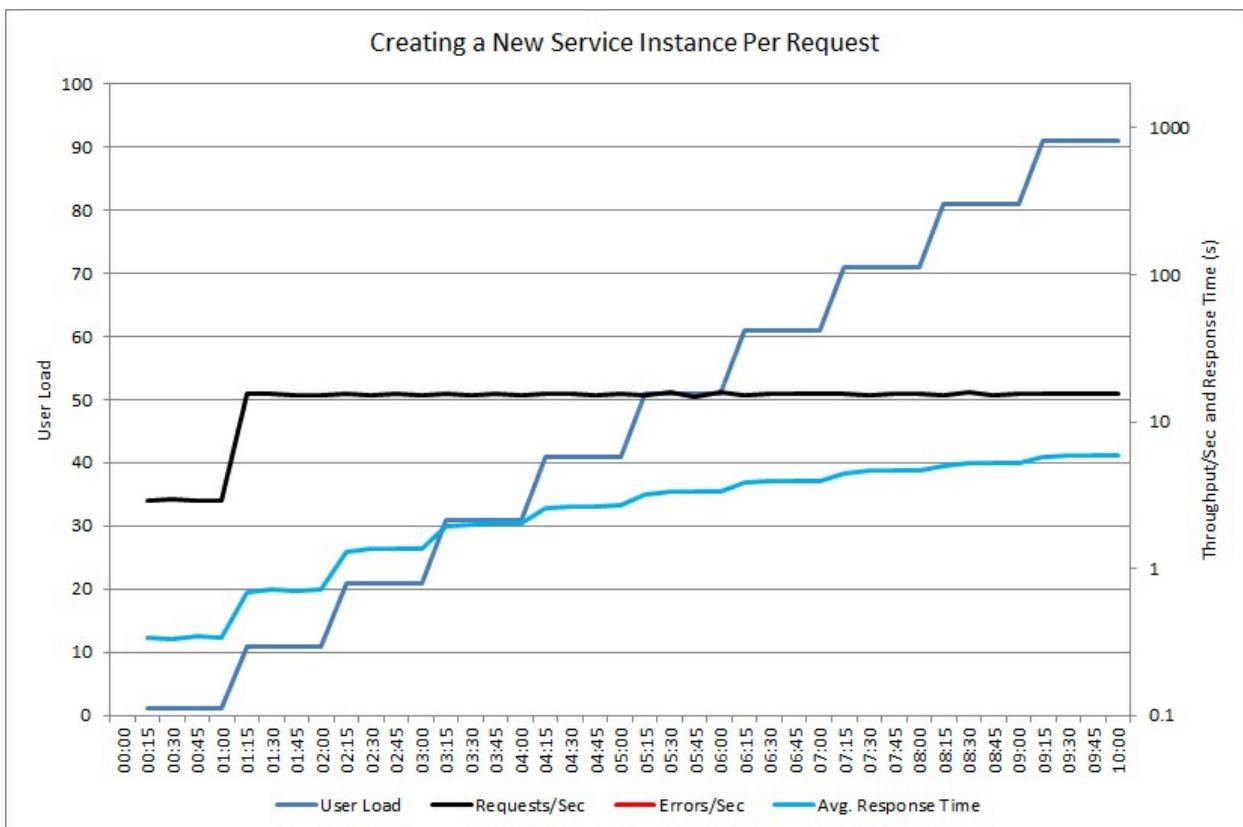
Use load testing to simulate the typical operations that users might perform. This can help to identify which parts of a system suffer from resource exhaustion under varying loads. Perform these tests in a controlled environment rather than the production system. The following graph shows the throughput of requests handled by the `NewHttpClientInstancePerRequest` controller as the user load increases to 100 concurrent users.



At first, the volume of requests handled per second increases as the workload increases. At about 30 users, however, the volume of successful requests reaches a limit, and the system starts to generate exceptions. From then on, the volume of exceptions gradually increases with the user load.

The load test reported these failures as HTTP 500 (Internal Server) errors. Reviewing the telemetry showed that these errors were caused by the system running out of socket resources, as more and more `HttpClient` objects were created.

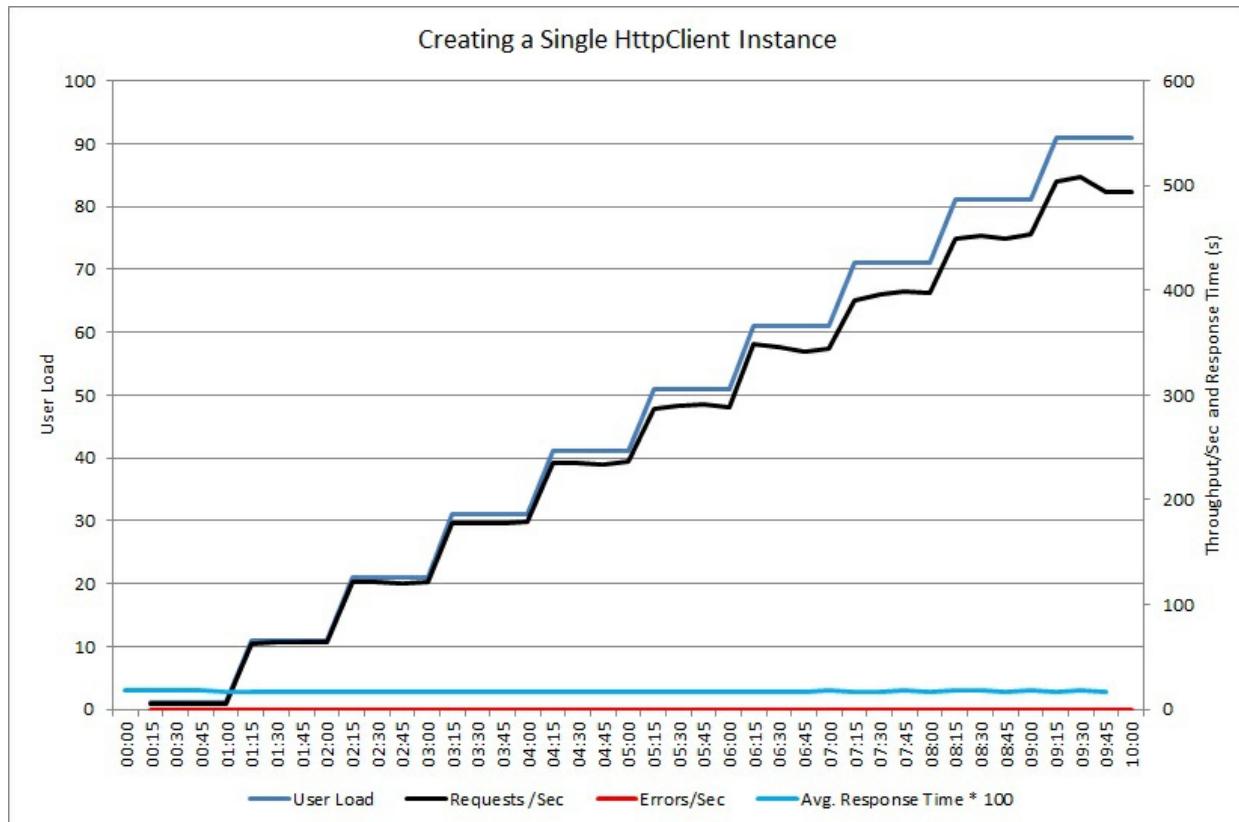
The next graph shows a similar test for a controller that creates the custom `ExpensiveToCreateService` object.



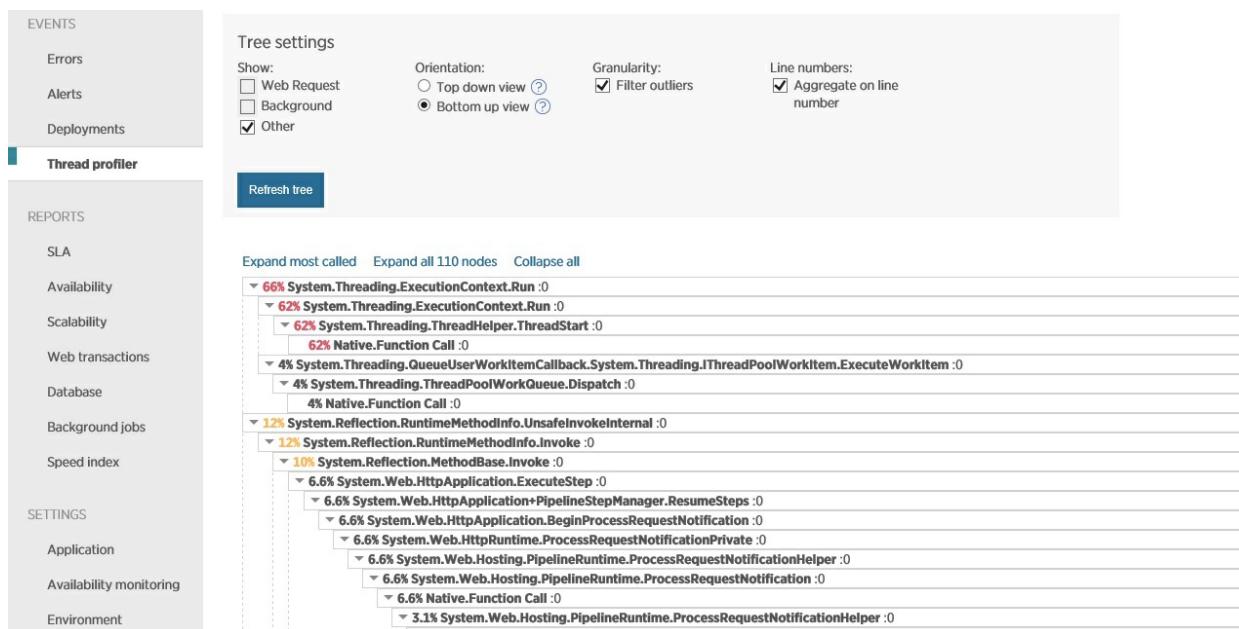
This time, the controller does not generate any exceptions, but throughput still reaches a plateau, while the average response time increases by a factor of 20. (The graph uses a logarithmic scale for response time and throughput.) Telemetry showed that creating new instances of the `ExpensiveToCreateService` was the main cause of the problem.

Implement the solution and verify the result

After switching the `GetProductAsync` method to share a single `HttpClient` instance, a second load test showed improved performance. No errors were reported, and the system was able to handle an increasing load of up to 500 requests per second. The average response time was cut in half, compared with the previous test.

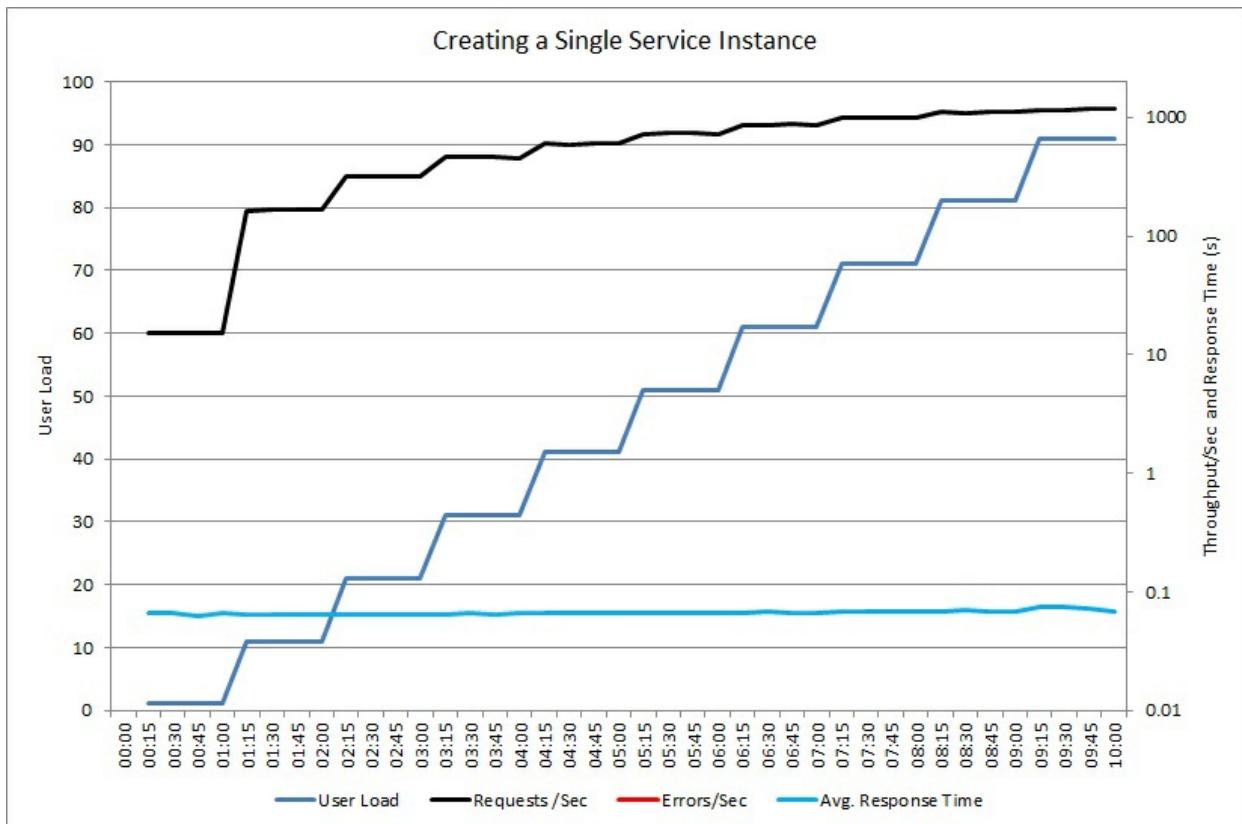


For comparison, the following image shows the stack trace telemetry. This time, the system spends most of its time performing real work, rather than opening and closing sockets.



The next graph shows a similar load test using a shared instance of the `ExpensiveToCreateService` object. Again, the volume of handled requests increases in line with the user load, while the average response time remains

low.



Monolithic Persistence antipattern

3/10/2022 • 6 minutes to read • [Edit Online](#)

Putting all of an application's data into a single data store can hurt performance, either because it leads to resource contention, or because the data store is not a good fit for some of the data.

Problem description

Historically, applications have often used a single data store, regardless of the different types of data that the application might need to store. Usually this was done to simplify the application design, or else to match the existing skill set of the development team.

Modern cloud-based systems often have additional functional and nonfunctional requirements, and need to store many heterogeneous types of data, such as documents, images, cached data, queued messages, application logs, and telemetry. Following the traditional approach and putting all of this information into the same data store can hurt performance, for two main reasons:

- Storing and retrieving large amounts of unrelated data in the same data store can cause contention, which in turn leads to slow response times and connection failures.
- Whichever data store is chosen, it might not be the best fit for all of the different types of data, or it might not be optimized for the operations that the application performs.

The following example shows an ASP.NET Web API controller that adds a new record to a database and also records the result to a log. The log is held in the same database as the business data. You can find the complete sample [here](#).

```
public class MonoController : ApiController
{
    private static readonly string ProductionDb = ...;

    public async Task<IHttpActionResult> PostAsync([FromBody]string value)
    {
        await DataAccess.InsertPurchaseOrderHeaderAsync(ProductionDb);
        await DataAccess.LogAsync(ProductionDb, LogTableName);
        return Ok();
    }
}
```

The rate at which log records are generated will probably affect the performance of the business operations. And if another component, such as an application process monitor, regularly reads and processes the log data, that can also affect the business operations.

How to fix the problem

Separate data according to its use. For each data set, select a data store that best matches how that data set will be used. In the previous example, the application should be logging to a separate store from the database that holds business data:

```

public class PolyController : ApiController
{
    private static readonly string ProductionDb = ...;
    private static readonly string LogDb = ...;

    public async Task<IHttpActionResult> PostAsync([FromBody]string value)
    {
        await DataAccess.InsertPurchaseOrderHeaderAsync(ProductionDb);
        // Log to a different data store.
        await DataAccess.LogAsync(LogDb, LogTableName);
        return Ok();
    }
}

```

Considerations

- Separate data by the way it is used and how it is accessed. For example, don't store log information and business data in the same data store. These types of data have significantly different requirements and patterns of access. Log records are inherently sequential, while business data is more likely to require random access, and is often relational.
- Consider the data access pattern for each type of data. For example, store formatted reports and documents in a document database such as [Cosmos DB](#), but use [Azure Cache for Redis](#) to cache temporary data.
- If you follow this guidance but still reach the limits of the database, you may need to scale up the database. Also consider scaling horizontally and partitioning the load across database servers. However, partitioning may require redesigning the application. For more information, see [Data partitioning](#).

How to detect the problem

The system will likely slow down dramatically and eventually fail, as the system runs out of resources such as database connections.

You can perform the following steps to help identify the cause.

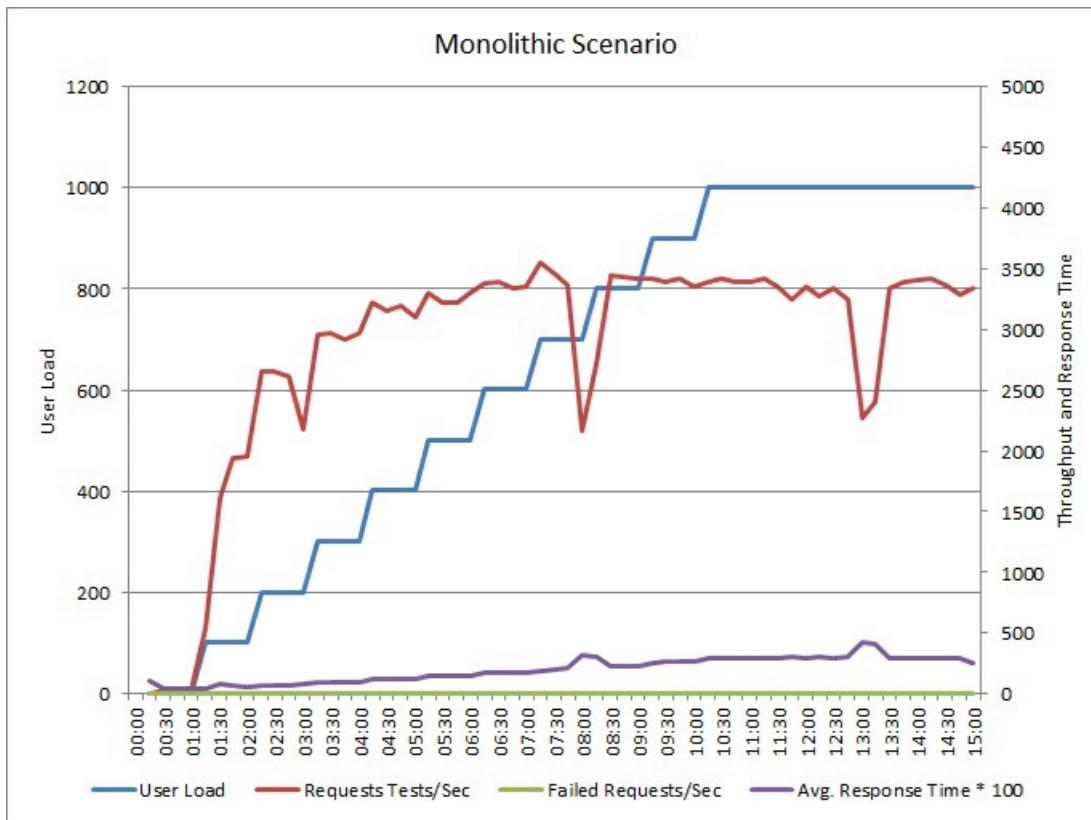
- Instrument the system to record the key performance statistics. Capture timing information for each operation, as well as the points where the application reads and writes data.
- If possible, monitor the system running for a few days in a production environment to get a real-world view of how the system is used. If this is not possible, run scripted load tests with a realistic volume of virtual users performing a typical series of operations.
- Use the telemetry data to identify periods of poor performance.
- Identify which data stores were accessed during those periods.
- Identify data storage resources that might be experiencing contention.

Example diagnosis

The following sections apply these steps to the sample application described earlier.

Instrument and monitor the system

The following graph shows the results of load testing the sample application described earlier. The test used a step load of up to 1000 concurrent users.



As the load increases to 700 users, so does the throughput. But at that point, throughput levels off, and the system appears to be running at its maximum capacity. The average response gradually increases with user load, showing that the system can't keep up with demand.

Identify periods of poor performance

If you are monitoring the production system, you might notice patterns. For example, response times might drop off significantly at the same time each day. This could be caused by a regular workload or scheduled batch job, or just because the system has more users at certain times. You should focus on the telemetry data for these events.

Look for correlations between increased response times and increased database activity or I/O to shared resources. If there are correlations, it means the database might be a bottleneck.

Identify which data stores are accessed during those periods

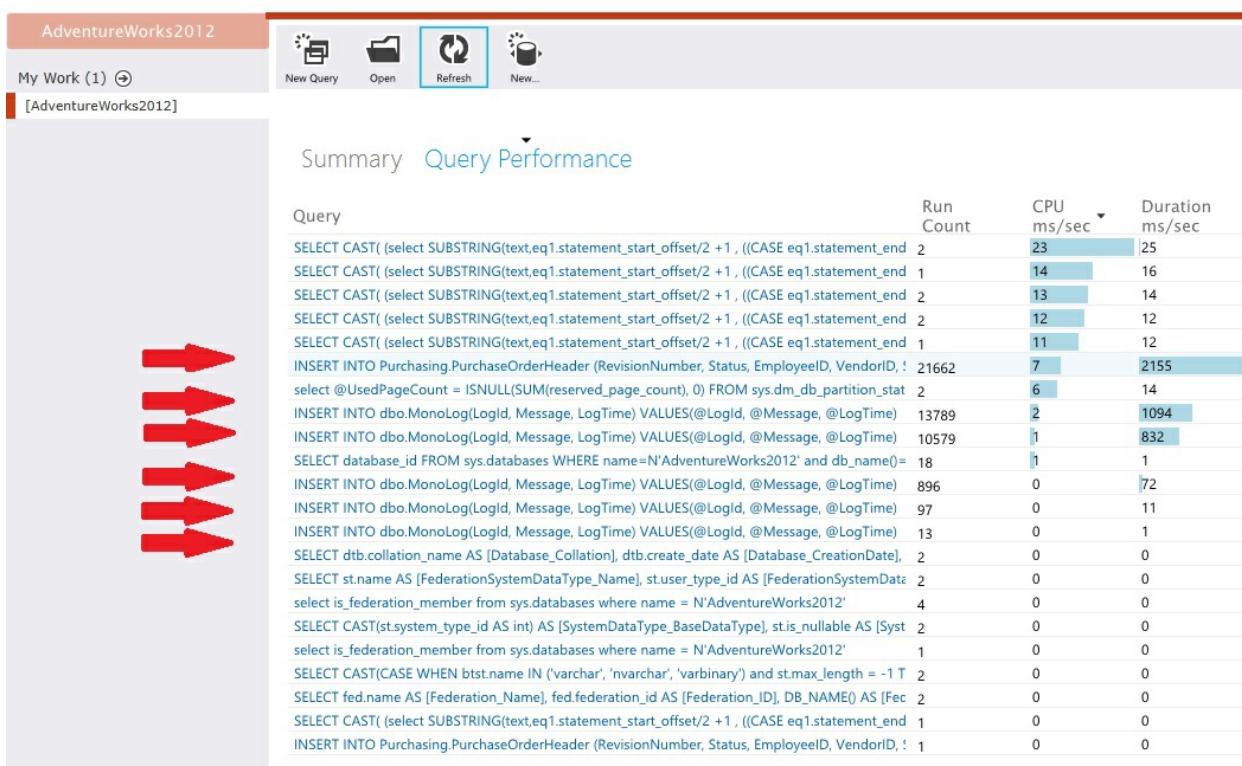
The next graph shows the utilization of database throughput units (DTU) during the load test. (A DTU is a measure of available capacity, and is a combination of CPU utilization, memory allocation, I/O rate.) Utilization of DTUs quickly reached 100%. This is roughly the point where throughput peaked in the previous graph. Database utilization remained very high until the test finished. There is a slight drop toward the end, which could be caused by throttling, competition for database connections, or other factors.

Monitoring



Examine the telemetry for the data stores

Instrument the data stores to capture the low-level details of the activity. In the sample application, the data access statistics showed a high volume of insert operations performed against both the `PurchaseOrderHeader` table and the `MonoLog` table.



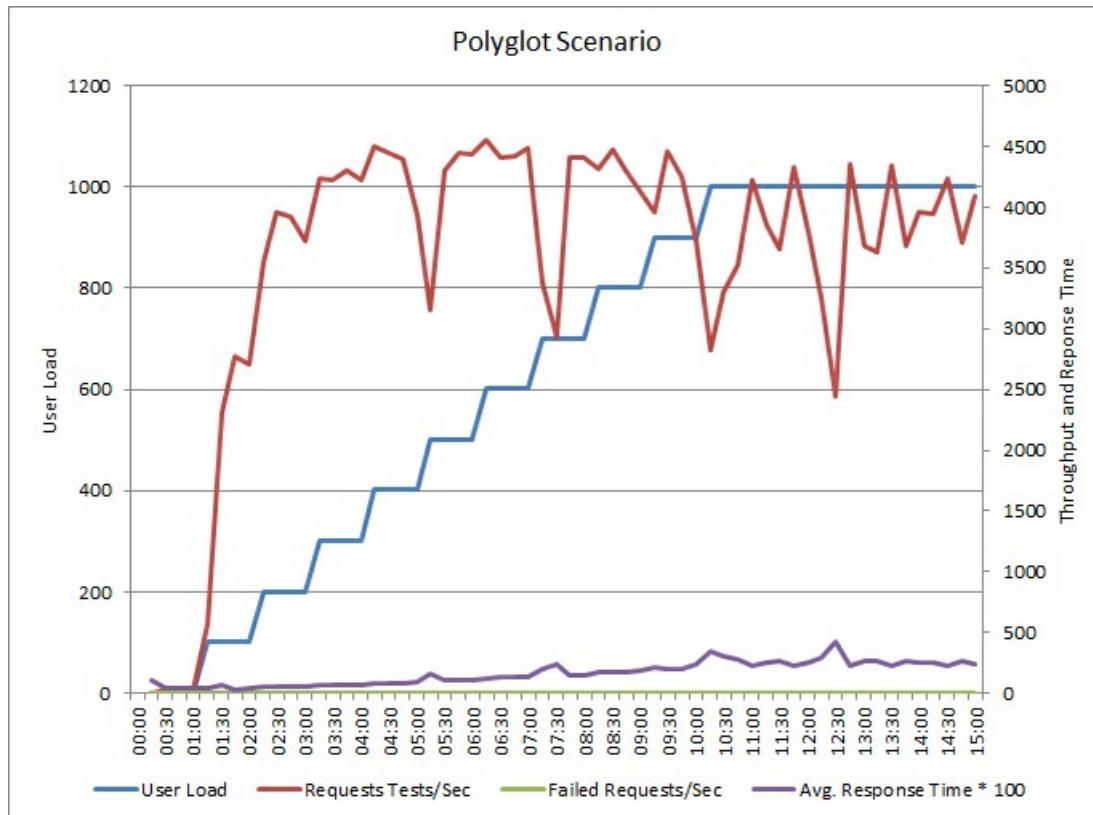
Identify resource contention

At this point, you can review the source code, focusing on the points where contended resources are accessed by the application. Look for situations such as:

- Data that is logically separate being written to the same store. Data such as logs, reports, and queued messages should not be held in the same database as business information.
- A mismatch between the choice of data store and the type of data, such as large blobs or XML documents in a relational database.
- Data with significantly different usage patterns that share the same store, such as high-write/low-read data being stored with low-write/high-read data.

Implement the solution and verify the result

The application was changed to write logs to a separate data store. Here are the load test results:



The pattern of throughput is similar to the earlier graph, but the point at which performance peaks is approximately 500 requests per second higher. The average response time is marginally lower. However, these statistics don't tell the full story. Telemetry for the business database shows that DTU utilization peaks at around 75%, rather than 100%.



Similarly, the maximum DTU utilization of the log database only reaches about 70%. The databases are no longer the limiting factor in the performance of the system.

Monitoring



Related resources

- [Choose the right data store](#)
- [Criteria for choosing a data store](#)
- [Data Access for Highly Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence](#)
- [Data partitioning](#)

No Caching antipattern

3/10/2022 • 8 minutes to read • [Edit Online](#)

Anti-patterns are common design flaws that can break your software or applications under stress situations and should not be overlooked. A *no caching antipattern* occurs when a cloud application that handles many concurrent requests, repeatedly fetches the same data. This can reduce performance and scalability.

When data is not cached, it can cause a number of undesirable behaviors, including:

- Repeatedly fetching the same information from a resource that is expensive to access, in terms of I/O overhead or latency.
- Repeatedly constructing the same objects or data structures for multiple requests.
- Making excessive calls to a remote service that has a service quota and throttles clients past a certain limit.

In turn, these problems can lead to poor response times, increased contention in the data store, and poor scalability.

Examples of no caching antipattern

The following example uses Entity Framework to connect to a database. Every client request results in a call to the database, even if multiple requests are fetching exactly the same data. The cost of repeated requests, in terms of I/O overhead and data access charges, can accumulate quickly.

```
public class PersonRepository : IPersonRepository
{
    public async Task<Person> GetAsync(int id)
    {
        using (var context = new AdventureWorksContext())
        {
            return await context.People
                .Where(p => p.Id == id)
                .FirstOrDefaultAsync()
                .ConfigureAwait(false);
        }
    }
}
```

You can find the complete sample [here](#).

This antipattern typically occurs because:

- Not using a cache is simpler to implement, and it works fine under low loads. Caching makes the code more complicated.
- The benefits and drawbacks of using a cache are not clearly understood.
- There is concern about the overhead of maintaining the accuracy and freshness of cached data.
- An application was migrated from an on-premises system, where network latency was not an issue, and the system ran on expensive high-performance hardware, so caching wasn't considered in the original design.
- Developers aren't aware that caching is a possibility in a given scenario. For example, developers may not think of using ETags when implementing a web API.

How to fix the no caching antipattern

The most popular caching strategy is the *on-demand* or *cache-aside* strategy.

- On read, the application tries to read the data from the cache. If the data isn't in the cache, the application retrieves it from the data source and adds it to the cache.
- On write, the application writes the change directly to the data source and removes the old value from the cache. It will be retrieved and added to the cache the next time it is required.

This approach is suitable for data that changes frequently. Here is the previous example updated to use the [Cache-Aside](#) pattern.

```
public class CachedPersonRepository : IPersonRepository
{
    private readonly PersonRepository _innerRepository;

    public CachedPersonRepository(PersonRepository innerRepository)
    {
        _innerRepository = innerRepository;
    }

    public async Task<Person> GetAsync(int id)
    {
        return await CacheService.GetAsync<Person>("p:" + id, () =>
_innerRepository.GetAsync(id)).ConfigureAwait(false);
    }
}

public class CacheService
{
    private static ConnectionMultiplexer _connection;

    public static async Task<T> GetAsync<T>(string key, Func<Task<T>> loadCache, double expirationTimeInMinutes)
    {
        IDatabase cache = Connection.GetDatabase();
        T value = await GetAsync<T>(cache, key).ConfigureAwait(false);
        if (value == null)
        {
            // Value was not found in the cache. Call the lambda to get the value from the database.
            value = await loadCache().ConfigureAwait(false);
            if (value != null)
            {
                // Add the value to the cache.
                await SetAsync(cache, key, value, expirationTimeInMinutes).ConfigureAwait(false);
            }
        }
        return value;
    }
}
```

Notice that the `GetAsync` method now calls the `CacheService` class, rather than calling the database directly. The `CacheService` class first tries to get the item from Azure Cache for Redis. If the value isn't found in the cache, the `CacheService` invokes a lambda function that was passed to it by the caller. The lambda function is responsible for fetching the data from the database. This implementation decouples the repository from the particular caching solution, and decouples the `CacheService` from the database.

Considerations for caching strategy

- If the cache is unavailable, perhaps because of a transient failure, don't return an error to the client. Instead, fetch the data from the original data source. However, be aware that while the cache is being recovered, the original data store could be swamped with requests, resulting in timeouts and failed connections. (After all, this is one of the motivations for using a cache in the first place.) Use a technique such as the [Circuit Breaker pattern](#) to avoid overwhelming the data source.

- Applications that cache dynamic data should be designed to support eventual consistency.
- For web APIs, you can support client-side caching by including a Cache-Control header in request and response messages, and using ETags to identify versions of objects. For more information, see [API implementation](#).
- You don't have to cache entire entities. If most of an entity is static but only a small piece changes frequently, cache the static elements and retrieve the dynamic elements from the data source. This approach can help to reduce the volume of I/O being performed against the data source.
- In some cases, if volatile data is short-lived, it can be useful to cache it. For example, consider a device that continually sends status updates. It might make sense to cache this information as it arrives, and not write it to a persistent store at all.
- To prevent data from becoming stale, many caching solutions support configurable expiration periods, so that data is automatically removed from the cache after a specified interval. You may need to tune the expiration time for your scenario. Data that is highly static can stay in the cache for longer periods than volatile data that may become stale quickly.
- If the caching solution doesn't provide built-in expiration, you may need to implement a background process that occasionally sweeps the cache, to prevent it from growing without limits.
- Besides caching data from an external data source, you can use caching to save the results of complex computations. Before you do that, however, instrument the application to determine whether the application is really CPU bound.
- It might be useful to prime the cache when the application starts. Populate the cache with the data that is most likely to be used.
- Always include instrumentation that detects cache hits and cache misses. Use this information to tune caching policies, such what data to cache, and how long to hold data in the cache before it expires.
- If the lack of caching is a bottleneck, then adding caching may increase the volume of requests so much that the web front end becomes overloaded. Clients may start to receive HTTP 503 (Service Unavailable) errors. These are an indication that you should scale out the front end.

How to detect a no caching antipattern

You can perform the following steps to help identify whether lack of caching is causing performance problems:

1. Review the application design. Take an inventory of all the data stores that the application uses. For each, determine whether the application is using a cache. If possible, determine how frequently the data changes. Good initial candidates for caching include data that changes slowly, and static reference data that is read frequently.
2. Instrument the application and monitor the live system to find out how frequently the application retrieves data or calculates information.
3. Profile the application in a test environment to capture low-level metrics about the overhead associated with data access operations or other frequently performed calculations.
4. Perform load testing in a test environment to identify how the system responds under a normal workload and under heavy load. Load testing should simulate the pattern of data access observed in the production environment using realistic workloads.
5. Examine the data access statistics for the underlying data stores and review how often the same data requests are repeated.

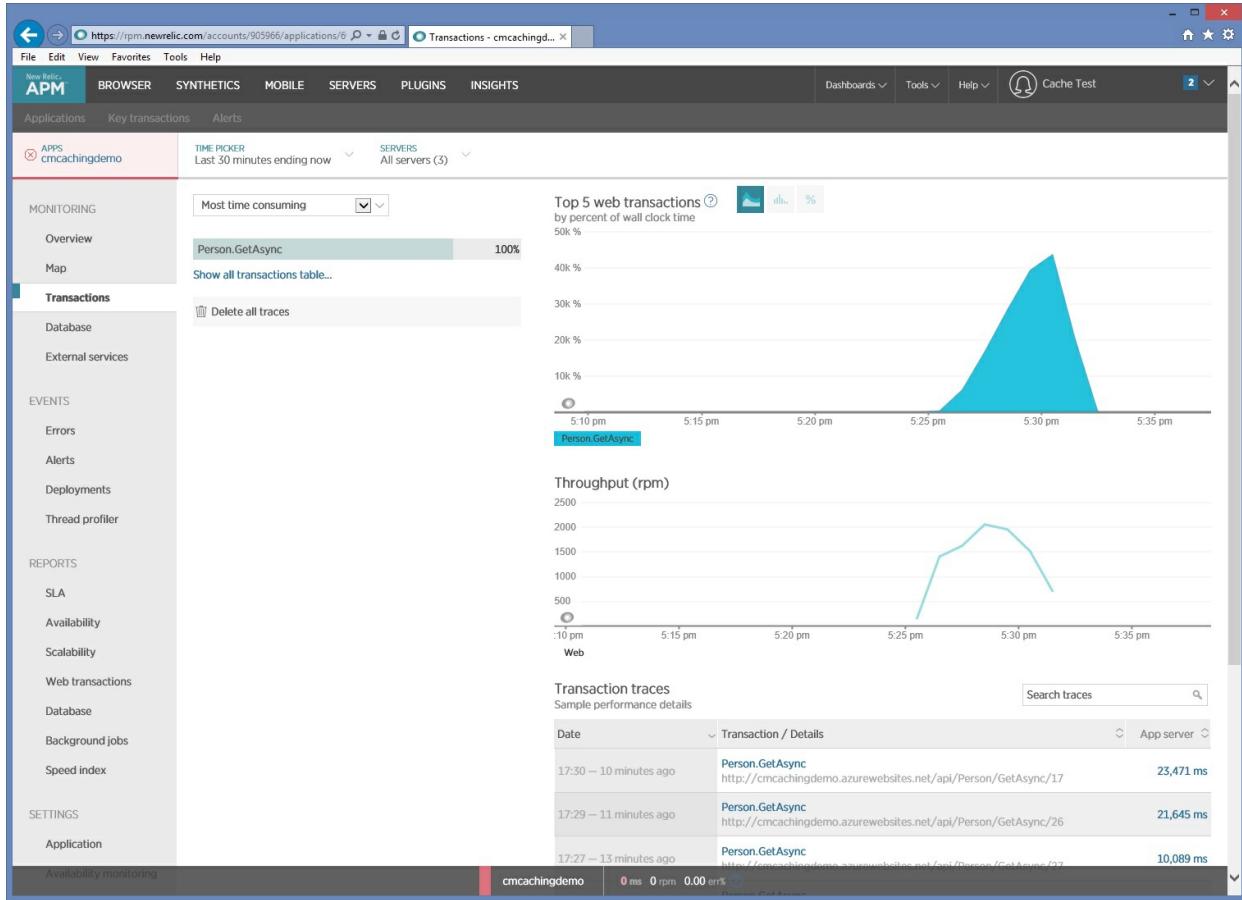
Example diagnosis

The following sections apply these steps to the sample application described earlier.

Instrument the application and monitor the live system

Instrument the application and monitor it to get information about the specific requests that users make while the application is in production.

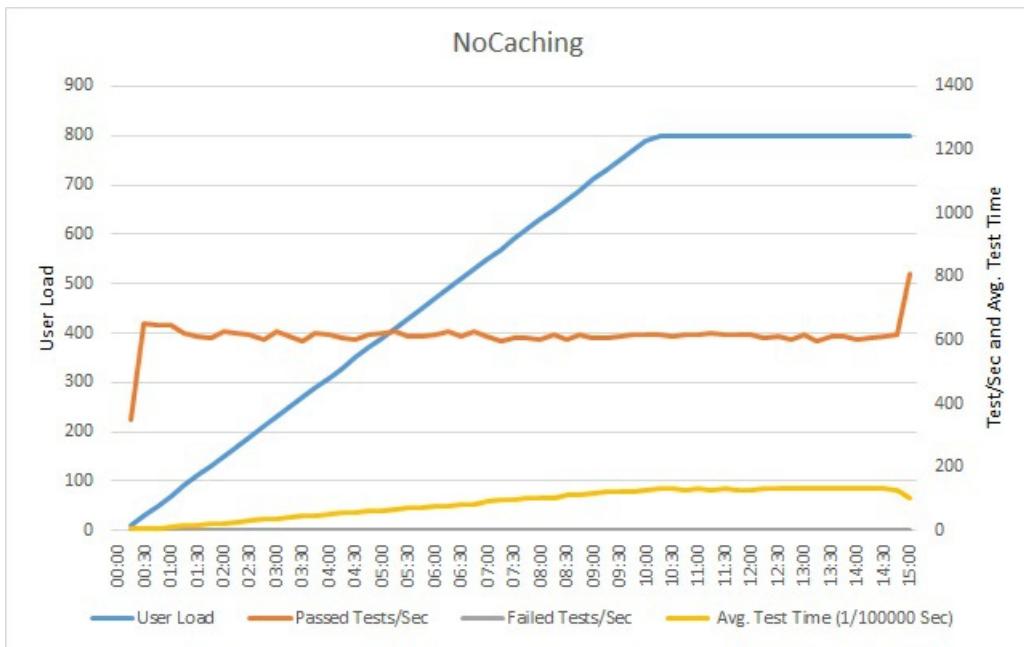
The following image shows monitoring data captured by [New Relic](#) during a load test. In this case, the only HTTP GET operation performed is `Person.GetAsync`. But in a live production environment, knowing the relative frequency that each request is performed can give you insight into which resources should be cached.



If you need a deeper analysis, you can use a profiler to capture low-level performance data in a test environment (not the production system). Look at metrics such as I/O request rates, memory usage, and CPU utilization. These metrics may show a large number of requests to a data store or service, or repeated processing that performs the same calculation.

Load test the application

The following graph shows the results of load testing the sample application. The load test simulates a step load of up to 800 users performing a typical series of operations.



The number of successful tests performed each second reaches a plateau, and additional requests are slowed as a result. The average test time steadily increases with the workload. The response time levels off once the user load peaks.

Examine data access statistics

Data access statistics and other information provided by a data store can give useful information, such as which queries are repeated most frequently. For example, in Microsoft SQL Server, the `sys.dm_exec_query_stats` management view has statistical information for recently executed queries. The text for each query is available in the `sys.dm_exec_query_plan` view. You can use a tool such as SQL Server Management Studio to run the following SQL query and determine how frequently queries are performed.

```
SELECT UseCounts, Text, Query_Plan
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
CROSS APPLY sys.dm_exec_query_plan(plan_handle)
```

The `useCount` column in the results indicates how frequently each query is run. The following image shows that the third query was run more than 250,000 times, significantly more than any other query.

```

SELECT UseCounts, Text, Query_Plan
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
CROSS APPLY sys.dm_exec_query_plan(plan_handle)

```

100 % <

Results Messages

UseCounts	Text	Query_Plan
1	SELECT UseCounts, Text, Query_Plan FROM sys.dm...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	select is_federation_member from sys.databases where ...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
3	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
4	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
5	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
6	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
7	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
8	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
9	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
10	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
11	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
12	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
13	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
14	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
15	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
16	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
17	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
18	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
19	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
20	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
21	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
22	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
23	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."
24	(@p__linq_0 int)SELECT TOP (2) [Extent1].[Busine...	<ShowPlanXML xmlns="http://schemas.microsoft.com..."

Here is the SQL query that is causing so many database requests:

```

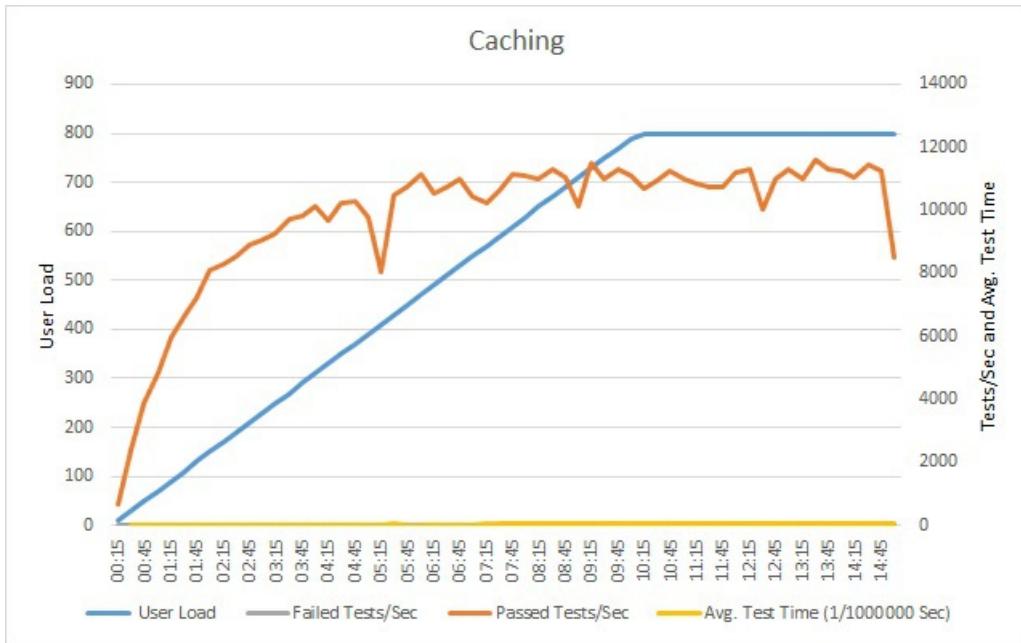
(@p__linq_0 int)SELECT TOP (2)
[Extent1].[BusinessEntityId] AS [BusinessEntityId],
[Extent1].[FirstName] AS [FirstName],
[Extent1].[LastName] AS [LastName]
FROM [Person].[Person] AS [Extent1]
WHERE [Extent1].[BusinessEntityId] = @p__linq_0

```

This is the query that Entity Framework generates in `GetByIdAsync` method shown earlier.

Implement the cache strategy solution and verify the result

After you incorporate a cache, repeat the load tests and compare the results to the earlier load tests without a cache. Here are the load test results after adding a cache to the sample application.



The volume of successful tests still reaches a plateau, but at a higher user load. The request rate at this load is significantly higher than earlier. Average test time still increases with load, but the maximum response time is 0.05 ms, compared with 1 ms earlier—a 20× improvement.

Related resources

- [API implementation best practices](#)
- [Cache-Aside pattern](#)
- [Caching best practices](#)
- [Circuit Breaker pattern](#)

Retry Storm antipattern

3/10/2022 • 5 minutes to read • [Edit Online](#)

When a service is unavailable or busy, having clients retry their connections too frequently can cause the service to struggle to recover, and can make the problem worse. It also doesn't make sense to retry forever, since requests are typically only valid for a defined period of time.

Problem description

In the cloud, services sometimes experience problems and become unavailable to clients, or have to throttle or rate limit their clients. While it's a good practice for clients to retry failed connections to services, it's important they do not retry too frequently or for too long. Retries within a short period of time are unlikely to succeed since the services likely will not have recovered. Also, services can be put under even more stress when lots of connection attempts are made while they're trying to recover, and repeated connection attempts may even overwhelm the service and make the underlying problem worse.

The following example illustrates a scenario where a client connects to a server-based API. If the request doesn't succeed, then the client retries immediately, and keeps retrying forever. Often this sort of behavior is more subtle than in this example, but the same principle applies.

```
public async Task<string> GetDataFromServer()
{
    while(true)
    {
        var result = await httpClient.GetAsync(string.Format("http://{0}:8080/api/...", hostName));
        if (result.IsSuccessStatusCode) break;
    }

    // ... Process result.
}
```

How to fix the problem

Client applications should follow some best practices to avoid causing a retry storm.

- Cap the number of retry attempts, and don't keep retrying for a long period of time. While it might seem easy to write a `while(true)` loop, you almost certainly don't want to actually retry for a long period of time, since the situation that led to the request being initiated has probably changed. In most applications, retrying for a few seconds or minutes is sufficient.
- Pause between retry attempts. If a service is unavailable, retrying immediately is unlikely to succeed. Gradually increase the amount of time you wait between attempts, for example by using an [exponential backoff strategy](#).
- Gracefully handle errors. If the service isn't responding, consider whether it makes sense to abort the attempt and return an error back to the user or caller of your component. Consider these failure scenarios when designing your application.
- Consider using the [Circuit Breaker pattern](#), which is designed specifically to help avoid retry storms.
- If the server provides a `retry-after` response header, make sure you don't attempt to retry until the specified time period has elapsed.
- Use official SDKs when communicating to Azure services. These SDKs generally have built-in retry policies and protections against causing or contributing to retry storms. If you're communicating with a service that

doesn't have an SDK, or where the SDK doesn't handle retry logic correctly, consider using a library like [Polly](#) (for .NET) or [retry](#) (for JavaScript) to handle your retry logic correctly and avoid writing the code yourself.

- If you're running in an environment that supports it, use a service mesh (or another abstraction layer) to send outbound calls. Typically these tools, such as [Dapr](#), support retry policies and automatically follow best practices, like backing off after repeated attempts. This approach means you don't have to write retry code yourself.
- Consider batching requests and using request pooling where available. Many SDKs handle request batching and connection pooling on your behalf, which will reduce the total number of outbound connection attempts your application makes, although you still need to be careful not to retry these connections too frequently.

Services should also protect themselves against retry storms.

- Add a gateway layer so you can shut off connections during an incident. This is an example of the [Bulkhead pattern](#). Azure provides many different gateway services for different types of solutions including [Front Door](#), [Application Gateway](#), and [API Management](#).
- Throttle requests at your gateway, which ensures you won't accept so many requests that your back-end components can't continue to operate.
- If you're throttling, send back a `retry-after` header to help clients understand when to reattempt their connections.

Considerations

- Clients should consider the type of error returned. Some error types don't indicate a failure of the service, but instead indicate that the client sent an invalid request. For example, if a client application receives a `400 Bad Request` error response, retrying the same request probably is not going to help since the server is telling you that your request is not valid.
- Clients should consider the length of time that makes sense to reattempt connections. The length of time you should retry for will be driven by your business requirements and whether you can reasonably propagate an error back to a user or caller. In most applications, retrying for a few seconds or minutes is sufficient.

How to detect the problem

From a client's perspective, symptoms of this problem could include very long response or processing times, along with telemetry that indicates repeated attempts to retry the connection.

From a service's perspective, symptoms of this problem could include a large number of requests from one client within a short period of time, or a large number of requests from a single client while recovering from outages. Symptoms could also include difficulty when recovering the service, or ongoing cascading failures of the service right after a fault has been repaired.

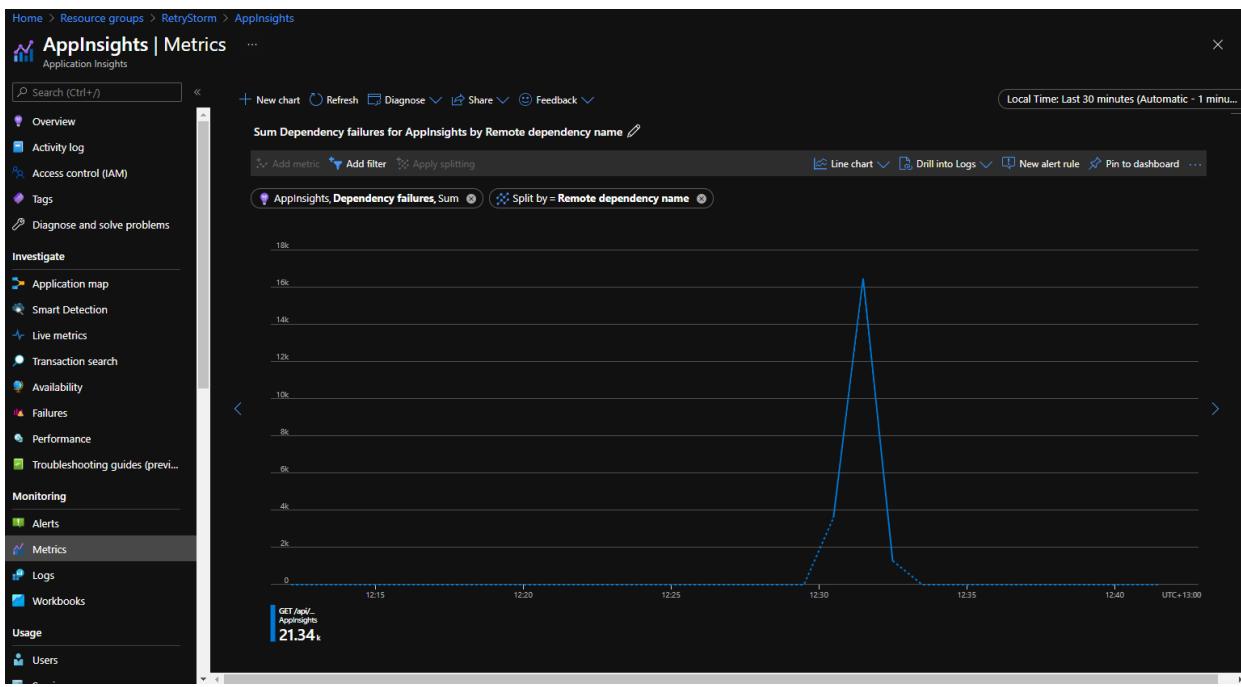
Example diagnosis

The following sections illustrate one approach to detecting a potential retry storm, both on the client side and the service side.

Identifying from client telemetry

[Azure Application Insights](#) records telemetry from applications and makes the data available for querying and visualization. Outbound connections are tracked as dependencies, and information about them can be accessed and charted to identify when a client is making a large number of outbound requests to the same service.

The following graph was taken from the Metrics tab within the Application Insights portal, and displaying the *Dependency failures* metric split by *Remote dependency name*. This illustrates a scenario where there were a large number (over 21,000) of failed connection attempts to a dependency within a short time.



Identifying from server telemetry

Server applications may be able to detect large numbers of connections from a single client. In the following example, Azure Front Door acts as a gateway for an application, and [has been configured to log](#) all requests to a Log Analytics workspace.

The following Kusto query can be executed against Log Analytics. It will identify client IP addresses that have sent large numbers of requests to the application within the last day.

```
AzureDiagnostics
| where ResourceType == "FRONDOORS" and Category == "FrontdoorAccessLog"
| where TimeGenerated > ago(1d)
| summarize count() by bin(TimeGenerated, 1h), clientIp_s
| order by count_ desc
```

Executing this query during a retry storm shows a large number of connection attempts from a single IP address.

The screenshot shows the Azure Log Analytics workspace. On the left, there's a navigation bar with "New Query 1*", "la-sgi767hzgug26", "Select scope", "Run", "Time range: Set in query", "Save", "Copy link", "New alert rule", "Export", and "Pin to dashboard". Below this is a table browser with "Tables" selected, showing "AzureDiagnostics" and a search bar. To the right is a query editor window with the previously shown Kusto query. At the bottom, the results pane shows a table with two rows:

TimeGenerated [UTC]	clientIp_s	count_
23/02/2021, 11:00:00.000 pm	[REDACTED]	81,608
23/02/2021, 11:00:00.000 pm	[REDACTED]	5

Related resources

- [Retry pattern](#)
- [Circuit Breaker pattern](#)
- [Transient fault handling best practices](#)
- [Service-specific retry guidance](#)

Synchronous I/O antipattern

3/10/2022 • 6 minutes to read • [Edit Online](#)

Blocking the calling thread while I/O completes can reduce performance and affect vertical scalability.

Problem description

A synchronous I/O operation blocks the calling thread while the I/O completes. The calling thread enters a wait state and is unable to perform useful work during this interval, wasting processing resources.

Common examples of I/O include:

- Retrieving or persisting data to a database or any type of persistent storage.
- Sending a request to a web service.
- Posting a message or retrieving a message from a queue.
- Writing to or reading from a local file.

This antipattern typically occurs because:

- It appears to be the most intuitive way to perform an operation.
- The application requires a response from a request.
- The application uses a library that only provides synchronous methods for I/O.
- An external library performs synchronous I/O operations internally. A single synchronous I/O call can block an entire call chain.

The following code uploads a file to Azure blob storage. There are two places where the code blocks waiting for synchronous I/O, the `CreateIfNotExists` method and the `UploadFromStream` method.

```
var blobClient = storageAccount.CreateCloudBlobClient();
var container = blobClient.GetContainerReference("uploadedfiles");

container.CreateIfNotExists();
var blockBlob = container.GetBlockBlobReference("myblob");

// Create or overwrite the "myblob" blob with contents from a local file.
using (var fileStream = File.OpenRead(HostingEnvironment.MapPath("~/FileToUpload.txt")))
{
    blockBlob.UploadFromStream(fileStream);
}
```

Here's an example of waiting for a response from an external service. The `GetUserProfile` method calls a remote service that returns a `UserProfile`.

```

public interface IUserProfileService
{
    UserProfile GetUserProfile();
}

public class SyncController : ApiController
{
    private readonly IUserProfileService _userProfileService;

    public SyncController()
    {
        _userProfileService = new FakeUserProfileService();
    }

    // This is a synchronous method that calls the synchronous GetUserProfile method.
    public UserProfile GetUserProfile()
    {
        return _userProfileService.GetUserProfile();
    }
}

```

You can find the complete code for both of these examples [here](#).

How to fix the problem

Replace synchronous I/O operations with asynchronous operations. This frees the current thread to continue performing meaningful work rather than blocking, and helps improve the utilization of compute resources. Performing I/O asynchronously is particularly efficient for handling an unexpected surge in requests from client applications.

Many libraries provide both synchronous and asynchronous versions of methods. Whenever possible, use the asynchronous versions. Here is the asynchronous version of the previous example that uploads a file to Azure blob storage.

```

var blobClient = storageAccount.CreateCloudBlobClient();
var container = blobClient.GetContainerReference("uploadedfiles");

await container.CreateIfNotExistsAsync();

var blockBlob = container.GetBlockBlobReference("myblob");

// Create or overwrite the "myblob" blob with contents from a local file.
using (var fileStream = File.OpenRead(HostingEnvironment.MapPath("~/FileToUpload.txt")))
{
    await blockBlob.UploadFromStreamAsync(fileStream);
}

```

The `await` operator returns control to the calling environment while the asynchronous operation is performed. The code after this statement acts as a continuation that runs when the asynchronous operation has completed.

A well designed service should also provide asynchronous operations. Here is an asynchronous version of the web service that returns user profiles. The `GetUserProfileAsync` method depends on having an asynchronous version of the User Profile service.

```

public interface IUserProfileService
{
    Task<UserProfile> GetUserProfileAsync();
}

public class AsyncController : ApiController
{
    private readonly IUserProfileService _userProfileService;

    public AsyncController()
    {
        _userProfileService = new FakeUserProfileService();
    }

    // This is an synchronous method that calls the Task based GetUserProfileAsync method.
    public Task<UserProfile> GetUserProfileAsync()
    {
        return _userProfileService.GetUserProfileAsync();
    }
}

```

For libraries that don't provide asynchronous versions of operations, it may be possible to create asynchronous wrappers around selected synchronous methods. Follow this approach with caution. While it may improve responsiveness on the thread that invokes the asynchronous wrapper, it actually consumes more resources. An extra thread may be created, and there is overhead associated with synchronizing the work done by this thread. Some tradeoffs are discussed in this blog post: [Should I expose asynchronous wrappers for synchronous methods?](#)

Here is an example of an asynchronous wrapper around a synchronous method.

```

// Asynchronous wrapper around synchronous library method
private async Task<int> LibraryIOOperationAsync()
{
    return await Task.Run(() => LibraryIOOperation());
}

```

Now the calling code can await on the wrapper:

```

// Invoke the asynchronous wrapper using a task
await LibraryIOOperationAsync();

```

Considerations

- I/O operations that are expected to be very short lived and are unlikely to cause contention might be more performant as synchronous operations. An example might be reading small files on an SSD drive. The overhead of dispatching a task to another thread, and synchronizing with that thread when the task completes, might outweigh the benefits of asynchronous I/O. However, these cases are relatively rare, and most I/O operations should be done asynchronously.
- Improving I/O performance may cause other parts of the system to become bottlenecks. For example, unblocking threads might result in a higher volume of concurrent requests to shared resources, leading in turn to resource starvation or throttling. If that becomes a problem, you might need to scale out the number of web servers or partition data stores to reduce contention.

How to detect the problem

For users, the application may seem unresponsive periodically. The application might fail with timeout

exceptions. These failures could also return HTTP 500 (Internal Server) errors. On the server, incoming client requests might be blocked until a thread becomes available, resulting in excessive request queue lengths, manifested as HTTP 503 (Service Unavailable) errors.

You can perform the following steps to help identify the problem:

1. Monitor the production system and determine whether blocked worker threads are constraining throughput.
2. If requests are being blocked due to lack of threads, review the application to determine which operations may be performing I/O synchronously.
3. Perform controlled load testing of each operation that is performing synchronous I/O, to find out whether those operations are affecting system performance.

Example diagnosis

The following sections apply these steps to the sample application described earlier.

Monitor web server performance

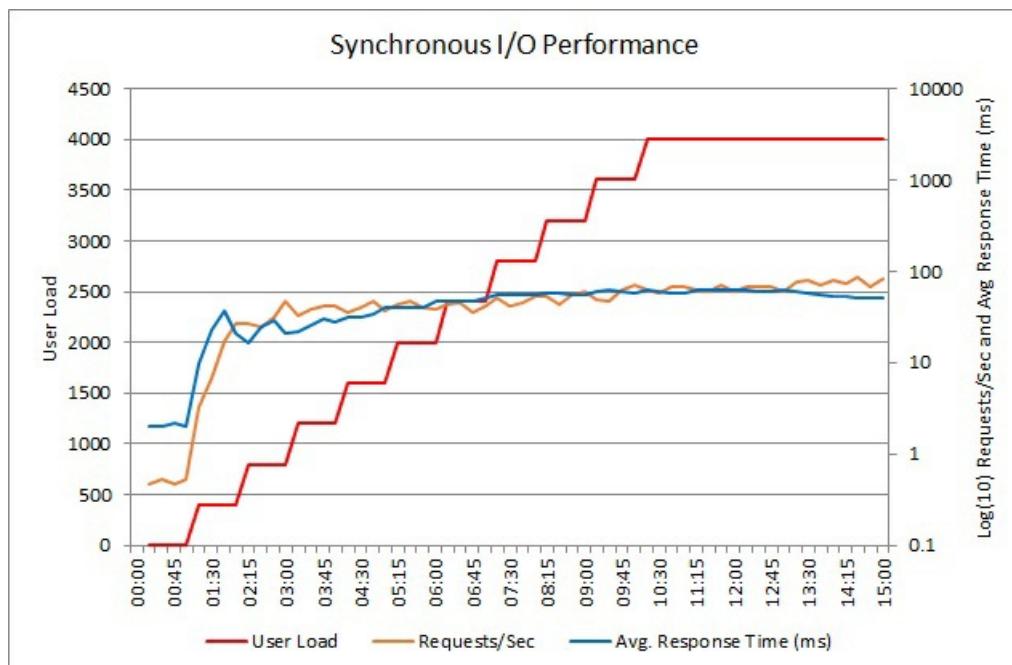
For Azure web applications and web roles, it's worth monitoring the performance of the IIS web server. In particular, pay attention to the request queue length to establish whether requests are being blocked waiting for available threads during periods of high activity. You can gather this information by enabling Azure diagnostics. For more information, see:

- [Monitor Apps in Azure App Service](#)
- [Create and use performance counters in an Azure application](#)

Instrument the application to see how requests are handled once they have been accepted. Tracing the flow of a request can help to identify whether it is performing slow-running calls and blocking the current thread. Thread profiling can also highlight requests that are being blocked.

Load test the application

The following graph shows the performance of the synchronous `GetUserProfile` method shown earlier, under varying loads of up to 4000 concurrent users. The application is an ASP.NET application running in an Azure Cloud Service web role.



The synchronous operation is hard-coded to sleep for 2 seconds, to simulate synchronous I/O, so the minimum

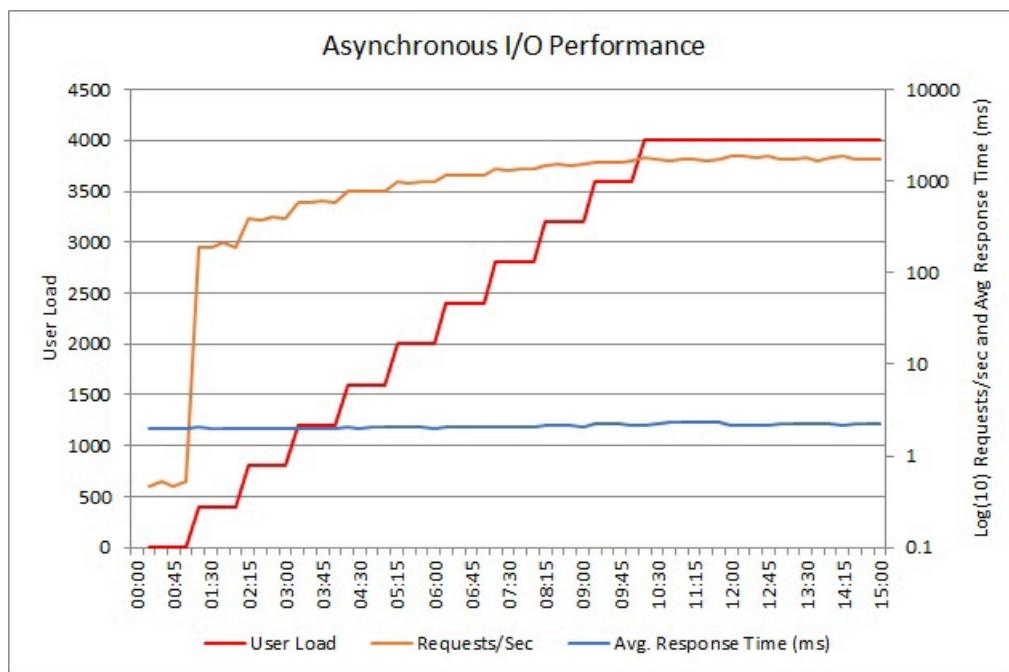
response time is slightly over 2 seconds. When the load reaches approximately 2500 concurrent users, the average response time reaches a plateau, although the volume of requests per second continues to increase. Note that the scale for these two measures is logarithmic. The number of requests per second doubles between this point and the end of the test.

In isolation, it's not necessarily clear from this test whether the synchronous I/O is a problem. Under heavier load, the application may reach a tipping point where the web server can no longer process requests in a timely manner, causing client applications to receive time-out exceptions.

Incoming requests are queued by the IIS web server and handed to a thread running in the ASP.NET thread pool. Because each operation performs I/O synchronously, the thread is blocked until the operation completes. As the workload increases, eventually all of the ASP.NET threads in the thread pool are allocated and blocked. At that point, any further incoming requests must wait in the queue for an available thread. As the queue length grows, requests start to time out.

Implement the solution and verify the result

The next graph shows the results from load testing the asynchronous version of the code.



Throughput is far higher. Over the same duration as the previous test, the system successfully handles a nearly tenfold increase in throughput, as measured in requests per second. Moreover, the average response time is relatively constant and remains approximately 25 times smaller than the previous test.

Responsible Innovation: A Best Practices Toolkit

3/10/2022 • 2 minutes to read • [Edit Online](#)

Responsible innovation is a toolkit that helps developers become good stewards for the future of science and its effect on society. This toolkit provides a set of practices in development, for anticipating and addressing the potential negative impacts of technology on people. We are sharing this as an early stage practice for feedback and learning.

Judgment Call

[Judgment Call](#) is an award-winning responsible innovation game and team-based activity that puts Microsoft's AI principles of fairness, privacy and security, reliability and safety, transparency, inclusion, and accountability into action. The game provides an easy-to-use method for cultivating stakeholder empathy through *scenario-imaging*. Game participants write product reviews from the perspective of a particular stakeholder, describing what kind of impact and harms the technology could produce from their point of view.

Harms Modeling

[Harms Modeling](#) is a framework for product teams, grounded in four core pillars of responsible innovation, that examine how people's lives can be negatively impacted by technology: injuries, denial of consequential services, infringement on human rights, and erosion of democratic & societal structures. Similar to *Security Threat Modeling*, Harms Modeling enables product teams to anticipate potential real-world impacts of technology, which is a cornerstone of responsible development.

Community Jury

[Community Jury](#) is a technique that brings together diverse stakeholders impacted by a technology. It is an adaptation of the [citizen jury](#). The stakeholders are provided an opportunity to learn from experts about a project, deliberate together, and give feedback on use cases and product design. This responsible innovation technique allows project teams to collaborate with researchers to identify stakeholder values, and understand the perceptions and concerns of impacted stakeholders.

Judgment Call

3/10/2022 • 5 minutes to read • [Edit Online](#)

Judgment Call is an award-winning game and team-based activity that puts Microsoft's AI principles of fairness, privacy and security, reliability and safety, transparency, inclusion, and accountability into action. The game provides an easy-to-use method for cultivating stakeholder empathy by imagining their scenarios. Game participants write product reviews from the perspective of a particular stakeholder, describing what kind of impact and harms the technology could produce from their point of view.

Benefits

Technology builders need practical methods to incorporate ethics in product development, by considering the values of diverse stakeholders and how technology may uphold or not uphold those values. The goal of the game is to imagine potential outcomes of a product or platform by gaining a better understanding of stakeholders, and what they need and expect.

The game helps people discuss challenging topics in a safe space within the context of gameplay, and gives technology creator a vocabulary to facilitate ethics discussions during product development. It gives managers and designers an interactive tool to lead ethical dialogue with their teams to incorporate ethical design in their products.

The theoretical basis and initial outcomes of the Judgment Call game were presented at the 2019 ACM Design Interactive Systems conference and the game was a finalist in the Fast Company 2019 Innovation by Design Awards (Social Goods category). The game has been presented to thousands of people in the US and internationally. While the largest group facilitated in game play has been over 100, each card deck can involve 1-10 players. This game is not intended to be a substitute for performing robust user research. Rather, it's an exercise to help tech builders learn how to exercise their moral imagination.

Preparation

Judgment Call uses role play to surface ethical concerns in product development so players will anticipate societal impact, write product reviews, and explore mitigations. Players think of what kind of harms and impacts the technology might produce by writing product reviews from the point of view of a stakeholder.

To prepare for this game, download the [printable Judgment Call game kit](#).

During the Game

The players are expected to go through the following steps in this game:

1. Identify the product

Start by identifying a product that your team is building, or a scenario you are working on. For example, if your team is developing synthetic voice technology, your scenario could be developing a voice assistant for online learning.

2. Pass the cards

Once you have decided on the scenario, pass out the cards. Each player receives a stakeholder card, an ethical principle card, a rating card, and a review form. The following is a description of these cards:

- **Stakeholder:** This represents the group of people impacted by the technology or scenario you've chosen. Stakeholders can also include the makers (those proposing and working on the technology) as well as the internal project sponsors (managers and executives who are supportive of the project).
- **Principle:** This includes Microsoft's ethical principles of fairness, privacy and security, reliability and safety, transparency, inclusion, and accountability.
- **Rating:** Star cards give a rating: 1-star is poor, 3-star is mediocre, 5-star is excellent.

3. Brainstorm and identify stakeholders

As a group, brainstorm all the stakeholders that could be directly or indirectly affected by the technology. Then assign each player a stakeholder from the list. In some cases, the Judgment Call decks may have pre-populated stakeholder cards, but players can add additional cards relevant to their situation.

The deck provides following categories of stakeholders:

- **Direct:** These use the technology and/or the technology is used on them.
- **Indirect:** These feel the effects of the technology.
- **Special populations:** A wide range of categories that includes those who cannot use the technology or choose not to use it, and organizations that may be interested in its deployment like advocacy groups or the government.

There are a range of harms that can be felt or perpetuated onto each group of stakeholder. Types of stakeholders can include end users of the product, administrators, internal teams, advocacy groups, manufacturers, someone who might be excluded from using the product. Stakeholders can also include the makers who propose and work on the technology, as well as the internal project sponsors, that is, the managers and executives who are supportive of the project.

4. Moderator selection

The moderator describes scenarios for gameplay, guides discussion, and collects comment cards at the end. They will also give an overview of the ethical principles considered in the deck.

5. Presentation of a product

The moderator introduces a product, real or imaginary, for the players to role play. They can start with questions about the product with prompts such as:

- Who will use the product?
- What features will it have?
- When will it be used?
- Where will it be used?
- Why is it useful?
- How will it be used?

6. Writing reviews

Players each write a review of the product or scenario based on the cards they've been dealt. Players take turns reading their reviews aloud. Listen carefully for any issues or themes that might come up. Take note of any changes you might make to your product based on insights discussed.

- The stakeholder card tells them who they are for the round.

- The ethical principle card is one of the six Microsoft AI principles; it tells them what area they're focusing on in their review.
- The rating card tells them how much their stakeholder likes the technology (1, 3, or 5 stars).

The player then writes a review of the technology, from the perspective of their stakeholder, of why they like or dislike the product from the perspective of the ethical principle they received.

Discussion

The moderator has each player read their reviews. Everyone is invited to discuss the different perspectives and other considerations that may not have come up.

Potential moderator questions include:

- Who is the most impacted?
- What features are problematic?
- What are the potential harms?

Harms mitigation

Finally, the group picks a thread from the discussion to begin identifying design or process changes that can mitigate a particular harm. At the end of each round, the decks are shuffled, and another round can begin with the same or a different scenario.

Next steps

Once you have enough understanding of potential harms or negative impact your product or scenario may cause, proceed to learn [how to model these harms](#) so you can devise effective mitigations.

Foundations of assessing harm

3/10/2022 • 3 minutes to read • [Edit Online](#)

Harms Modeling is a practice designed to help you anticipate the potential for harm, identify gaps in product that could put people at risk, and ultimately create approaches that proactively address harm.

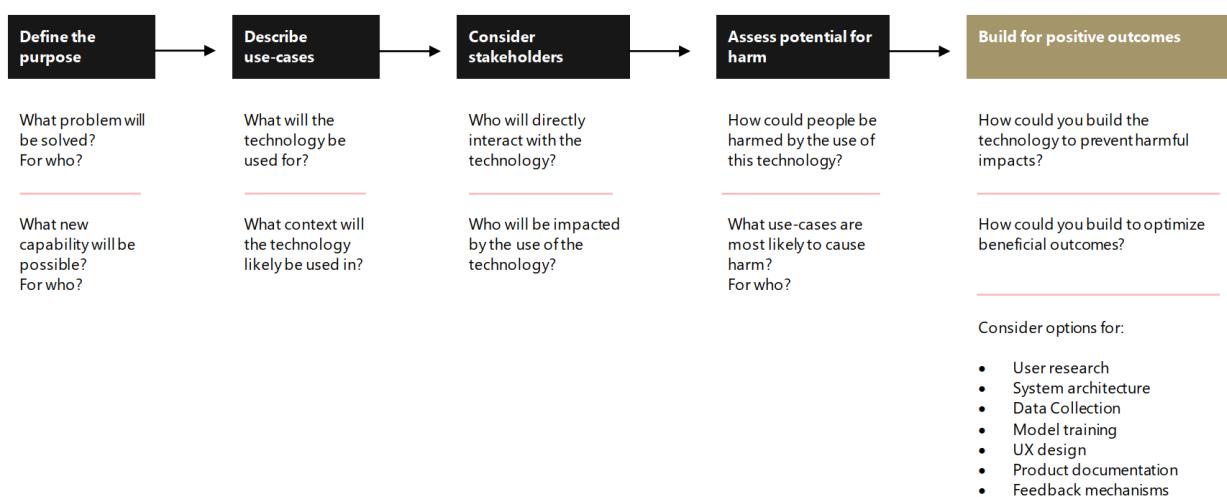
Why harms modeling?

As technology builders, your work is global. Designing AI to be trustworthy requires creating solutions that reflect ethical principles deeply rooted in important and timeless values. In the process of designing and building your technologies, it is essential to evaluate not only ideal outcomes, but possible negative ones as well.

Technology and human rights

It is as important as ever to be aware of how digital technology could impact human rights. In addition to continuing to protect privacy and security, we must address the risks of AI and other emerging technologies, such as facial recognition. History teaches us that human rights violations not only result from the nefarious use of technology, but also from a lack of awareness amongst those who have good intentions. As a part of our company's dedication to the protection of human rights, Microsoft forged a partnership with important stakeholders outside of our industry, including the United Nations (UN).

An important set of UN principles that our company firmly supports, which was ratified by over 250 nations, is the Universal Declaration of Human Rights (UDHR). The UDHR is a milestone document. Drafted by a diverse global group of legal and cultural experts, the Declaration was proclaimed by the United Nations General Assembly in 1948 as a common standard of achievements for all peoples and all nations. It sets out, for the first time, fundamental human rights to be universally protected. It has been translated into over 500 languages. Additionally, Microsoft is one of 4,700 corporate signatories to the UN Global Compact, an international business initiative designed to promote responsible corporate citizenship.



[Download in Microsoft Word](#)

Human understanding

In addition to appreciating the importance of human rights, building trustworthy systems requires us to consider many people's perspectives. Asking who the stakeholders are, what they value, how they could benefit, and how they could be hurt by our technology, is a powerful step that allows us to design and build better products.

Who does the technology impact?

Who are the customers?

- What do they value?
- How should they benefit?
- How could tech harm them?

Who are the non-customer stakeholders?

- What do they value?
- How should they benefit?
- How could tech harm them?

Asking these questions is a practice in [Value Sensitive Design](#) and is the beginning to better understanding what is important to stakeholders, and how it plays into their relationship with the product.

Types of stakeholders

Project sponsors

Backers, decision makers, and owners make up this category. Their values are articulated in the project strategy and goals.

Tech builders

Designers, developers, project managers, and people working directly on designing systems make up this group. They bring their own ethical standards and profession-specific values to the system.

Direct & indirect stakeholders

These stakeholders are significantly impacted by the system. Those who are impacted include end users, software staff, clients, bystanders, interfacing institutions, and even past or future generations. Non-human factors such as places, for example, historic buildings or sacred spaces, may also be included.

Marginalized populations

This category is made up of the population frequently considered a minority, vulnerable, or stigmatized. This category includes children, the elderly, members of the LGBTQ+ community, ethnic minorities, and other populations who often experience unique and disproportionate consequences.

Assessing Harm

Once you have defined the technology purpose, use cases, and stakeholders, conduct a Harms Modeling exercise to evaluate potential ways the use of a technology you are building could result in negative outcomes for people and society.

CATEGORY	TYPE OF HARM	Severity	Scale	Probability	Frequency	POTENTIAL
Risk of injury	Physical or infrastructure damage	■	■	■	■	LOW
	Emotional or psychological distress	▲	■	▲	▲	HIGH
Denial of consequential services	Opportunity loss	■	■	■	■	LOW
	Economic loss	■	■	■	■	LOW
Infringement on human rights	Dignity loss	■	■	■	■	MODERATE
	Liberty loss	■	■	■	■	LOW
	Privacy loss	▲	■	▲	▲	HIGH
	Environmental impact	■	■	■	■	LOW
Erosion of social & democratic structures	Manipulation	▲	■	■	▲	HIGH
	Social detriment	■	■	■	■	MODERATE

NOTE: This summary represents the outcome of a qualitative assessment and is used to inform prioritization of responsible innovation mitigations.

[Download in Microsoft Word](#)

The diagram above is an example of a harms evaluation. This model is a qualitative approach that's used to understand the potential magnitude of harm.

You can complete this ideation activity individually, but ideally it is conducted as collaboration between developers, data scientists, designers, user researcher, business decision-makers, and other disciplines that are involved in building the technology.

Suggestions for harm description statements:

- Intended use: If [feature] was used for [use case], then [stakeholder] could experience [harm description].
- Unintended use: If [user] tried to use [feature] for [use case], then [stakeholder] could experience [harm description].
- System error: If [feature] failed to function properly when used for [use case], then [stakeholder] could experience [harm description].
- Misuse: [Malicious actor] could potentially use [feature], to cause [harm description] to [stakeholder].

Use the categories, questions, and examples described in the [Types of harm](#) to generate specific ideas for how harm could occur. The article lists categories of harms, that are based upon common negative impact areas. Adapt and adopt additional categories that are relevant to you.

Next Steps

Read [Types of harm](#) for further harms analysis.

Types of harm

3/10/2022 • 13 minutes to read • [Edit Online](#)

This article creates awareness for the different types of harms, so that appropriate mitigation steps can be implemented.

Risk of injury

Physical injury

Consider how technology could hurt people or create dangerous environments.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Overreliance on safety features	This points out the dependence on technology to make decisions without adequate human oversight.	How might people rely on this technology to keep them safe? How could this technology reduce appropriate human oversight?	A healthcare agent could misdiagnose illness, leading to unnecessary treatment.
Inadequate fail-safes	Real-world testing may insufficiently consider a diverse set of users and scenarios.	If this technology fails or is misused, how would people be impacted? At what point could a human intervene? Are there alternative uses that have not been tested for? How would users be impacted by a system failure?	If an automatic door failed to detect a wheelchair during an emergency evacuation, a person could be trapped if there isn't an accessible override button.
Exposure to unhealthy agents	Manufacturing, as well as disposal of technology can jeopardize the health and well-being of workers and nearby inhabitants.	What negative outcomes could come from the manufacturing of your components or devices?	Inadequate safety measures could cause workers to be exposed to toxins during digital component manufacturing.

Emotional or psychological injury

Misused technology can lead to serious emotional and psychological distress.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Overreliance on automation	Misguided beliefs can lead users to trust the reliability of a digital agent over that of a human.	How could this technology reduce direct interpersonal feedback? How might this technology interface with trusted sources of information? How could sole dependence on an artificial agent impact a person?	A chat bot could be relied upon for relationship advice or mental health counseling instead of a trained professional.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Distortion of reality or gaslighting	When intentionally misused, technology can undermine trust and distort someone's sense of reality.	Could this be used to modify digital media or physical environments?	An IoT device could enable monitoring and controlling of an ex-intimate partner from afar.
Reduced self-esteem/reputation damage	Some shared content can be harmful, false, misleading, or denigrating.	How could this technology be used to inappropriately share personal information? How could it be manipulated to misuse information and misrepresent people?	Synthetic media "revenge porn" can swap faces, creating the illusion of a person participating in a video, who did not.
Addiction/attention hijacking	Technology could be designed for prolonged interaction, without regard for well-being.	In what ways might this technology reward or encourage continued interaction beyond delivering user value?	Variable drop rates in video game loot boxes could cause players to keep playing and neglect self-care.
Identity theft	Identity theft may lead to loss of control over personal credentials, reputation, and/or representation.	How might an individual be impersonated with this technology? How might this technology mistakenly recognize the wrong individual as an authentic user?	Synthetic voice font could mimic the sound of a person's voice and be used to access a bank account.
Misattribution	This includes crediting a person with an action or content that they are not responsible for.	In what ways might this technology attribute an action to an individual or group? How could someone be affected if an action was incorrectly attributed to them?	Facial recognition can misidentify an individual during a police investigation.

Denial of consequential services

Opportunity loss

Automated decisions could limit access to resources, services, and opportunities essential to well-being.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Employment discrimination	Some people may be denied access to apply for or secure a job based on characteristics unrelated to merit.	Are there ways in which this technology could impact recommendations or decisions related to employment?	Hiring AI could recommend fewer candidates with female-sounding names for interviews.
Housing discrimination	This includes denying people access to housing or the ability to apply for housing.	How could this technology impact recommendations or decisions related to housing?	Public housing queuing algorithm could cause people with international-sounding names to wait longer for vouchers.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Insurance and benefit discrimination	This includes denying people insurance, social assistance, or access to a medical trial due to biased standards.	Could this technology be used to determine access, cost, allocation of insurance or social benefits?	Insurance company might charge higher rates for drivers working night shifts due to algorithmic predictions suggesting increased drunk driving risk.
Educational discrimination	Access to education may be denied because of an unchangeable characteristic.	How might this technology be used to determine access, cost, accommodations, or other outcomes related to education?	Emotion classifier could incorrectly report that students of color are less engaged than their white counterparts, leading to lower grades.
Digital divide/technological discrimination	Disproportionate access to the benefits of technology, may leave some people less informed or less equipped to participate in society.	What prerequisite skills, equipment, or connectivity are necessary to get the most out of this technology? What might be the impact of select people gaining earlier access to this technology than others, in terms of equipment, connectivity, or other product functionality?	Content throttling could prevent rural students from accessing classroom instruction video feed.
Loss of choice/network and filter bubble	Presenting people with only information that conforms to and reinforces their own beliefs.	How might this technology affect which choices and information are made available to people? What past behaviors or preferences might this technology rely on to predict future behaviors or preferences?	News feed could only present information that confirms existing beliefs.

Economic loss

Automating decisions related to financial instruments, economic opportunity, and resources can amplify existing societal inequities and obstruct well-being.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Credit discrimination	People may be denied access to financial instruments based on characteristics unrelated to economic merit.	How might this technology rely on existing credit structures to make decisions? How might this technology affect the ability of an individual or group to obtain or maintain a credit score?	Higher introductory rate offers could be sent only to homes in lower socioeconomic postal codes.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Differential pricing of goods and services	Goods or services may be offered at different prices for reasons unrelated to the cost of production or delivery.	How could this technology be used to determine pricing of goods or services? What are the criteria for determining the cost to people for use of this tech?	More could be charged for products based on designation for men or women.
Economic exploitation	People might be compelled or misled to work on something that impacts their dignity or wellbeing.	What role did human labor play in producing training data for this technology? How was this workforce acquired? What role does human labor play in supporting this technology? Where is this workforce expected to come from?	Paying financially destitute people for their biometric data to train AI systems.
Devaluation of individual expertise	Technology may supplant the use of paid human expertise or labor.	How might this technology impact the need to employ an existing workforce?	AI agents replace doctors/radiographers for evaluation of medical imaging.

Infringement on human rights

Dignity loss

Technology can influence how people perceive the world, and how they recognize, engage, and value one another. The exchange of honor and respect between people can be interfered with.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Dehumanization	Removing, reducing, or obscuring visibility of a person's humanity.	How might this technology be used to simplify or abstract the way a person is represented? How might this technology reduce the distinction between humans and the digital world?	Entity recognition and virtual overlays in drone surveillance could reduce the perceived accountability of human actions.
Public shaming	This may mean exposing people's private, sensitive, or socially inappropriate material.	How might movements or actions be revealed through data aggregation?	A fitness app could reveal a user's GPS location on social media, indicating attendance at an Alcoholics Anonymous meeting.

Liberty loss

Automation of legal, judicial, and social systems can reinforce biases and lead to detrimental consequences.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Predictive policing	Inferring suspicious behavior and/or criminal intent based on historical records.	How could this support or replace human policing or criminal justice decision-making?	An algorithm can predict a number of area arrests, so police make sure they match, or exceed, that number.
Social control	Conformity may be reinforced or encouraged by publicly designating human behaviors as positive or negative.	What types of personal or behavioral data might feed this technology? How would it be obtained? What outputs would be derived from this data? Is this technology likely to be used to encourage or discourage certain behaviors?	Authoritarian government uses social media and e-commerce data to determine a "trustworthy" score based on where people shop and who they spend time with.
Loss of effective remedy	This means an inability to explain the rationale or lack of opportunity to contest a decision.	How might people understand the reasoning for decisions made by this technology? How might an individual that relies on this technology explain the decisions it makes? How could people contest or question a decision this technology makes?	Automated prison sentence or pre-trial release decision is not explained to the accused.

Privacy loss

Information generated by our use of technology can be used to determine facts or make assumptions about someone without their knowledge.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Interference with private life	Revealing information a person has not chosen to share.	How could this technology use information to infer portions of a private life? How could decisions based upon these inferences expose things that a person does not want made public?	Task-tracking AI could monitor personal patterns from which it infers an extramarital affair.
Forced association	Requiring participation in the use of technology or surveillance to take part in society.	How might use of this technology be required for participation in society or organization membership?	Biometric enrollment in a company's meeting room transcription AI is a stipulated requirement in job offer letter.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Inability to freely and fully develop personality	This may mean restriction of one's ability to truthfully express themselves or explore external avenues for self-development.	In what way does the system or product ascribe positive vs negative connotations toward particular personality traits? In what way can using the product or system reveal information to entities such as the government or employer that inhibits free expression?	Intelligent meeting system could record all discussions between colleagues including personal coaching and mentorship sessions.
Never forgiven	Digital files or records may never be deleted.	What and where is data being stored from this product, and who can access it? How long is user data stored after technology interaction? How is user data updated or deleted?	A teenager's social media history could remain searchable long after they have outgrown the platform.
Loss of freedom of movement or assembly	This means an inability to navigate the physical or virtual world with desired anonymity.	In what ways might this technology monitor people across physical and virtual space?	A real name could be required in order to sign up for a video game enabling real-world stalking.

Environmental impact

The environment can be impacted by every decision in a system or product life cycle, from the amount of cloud computing needed to retail packaging. Environmental changes can impact entire communities.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Exploitation or depletion of resources	Obtaining the raw materials for a technology, including how it's powered, leads to negative consequences to the environment and its inhabitants.	What materials are needed to build or run this technology? What energy requirements are needed to build or run this technology?	A local community could be displaced due to the harvesting of rare earth minerals and metals required for some electronic manufacturing.
Electronic waste	Reduced quality of collective well-being because of the inability to repair, recycle, or otherwise responsibly dispose of electronics.	How might this technology reduce electronic waste by recycling materials or allowing users to self-repair? How might this technology contribute to electronic waste when new versions are released or when current/past versions stop working?	Toxic materials inside discarded electronic devices could leach into the water supply, making local populations ill.

Erosion of social & democratic structures

Manipulation

The ability for technology to be used to create highly personalized and manipulative experiences can undermine an informed citizenry and trust in societal structures.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Misinformation	Disguising fake information as legitimate or credible information.	How might this technology be used to generate misinformation? How could it be used to spread misinformation that appears credible?	Generation of synthetic speech of a political leader sways an election.
Behavioral exploitation	This means exploiting personal preferences or patterns of behavior to induce a desired reaction.	How might this technology be used to observe patterns of behavior? How could this technology be used to encourage dysfunctional or maladaptive behaviors?	Monitoring shopping habits in connected retail environment leads to personalized incentives for impulse shoppers and hoarders.

Social detriment

At scale, the way technology impacts people shapes social and economic structures within communities. It can further ingrain elements that include or benefit some, at the exclusion of others.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Amplification of power inequality	This may perpetuate existing class or privilege disparities.	How might this technology be used in contexts where there are existing social, economic, or class disparities? How might people with more power or privilege disproportionately influence the technology?	Requiring a residential address & phone number to register on a job website could prevent a homeless person from applying for jobs.
Stereotype reinforcement	This may perpetuate uninformed "conventional wisdom" about historically or statistically underrepresented people.	How might this technology be used to reinforce or amplify existing social norms or cultural stereotypes? How might the data used by this technology cause it to reflect biases or stereotypes?	Results of an image search for "CEO" could primarily show photos of Caucasian men.
Loss of individuality	This may be an inability to express a unique perspective.	How might this technology amplify majority opinions or "group-think"? Conversely, how might unique forms of expression be suppressed. In what ways might the data gathered by this technology be used in feedback to people?	Limited customization options in designing a video game avatar inhibits self-expression of a player's diversity.

HARM	DESCRIPTION	CONSIDERATION(S)	EXAMPLE
Loss of representation	Broad categories of generalization obscure, diminish, or erase real identities.	How could this technology constrain identity options? Could it be used to automatically label or categorize people?	Automatic photo caption assigns incorrect gender identity and age to the subject.
Skill degradation and complacency	Overreliance on automation leads to atrophy of manual skills.	In what ways might this technology reduce the accessibility and ability to use manual controls?	Overreliance on automation could lead to an inability to gauge the airplane's true orientation because the pilots have been trained to rely on instruments only.

Evaluate harms

Once you have generated a broad list of potential harms, you should complete your Harms Model by evaluating the potential magnitude for each category of harm. This will allow you to prioritize your areas of focus. See the following example harms model for reference:

CONTRIBUTING FACTOR	DEFINITION
Severity	How acutely could an individual or group's well-being be impacted by the technology?
Scale	How broadly could the impact to well-being be experienced across populations or groups?
Probability	How likely is it that individual or group's well-being will be impacted by the technology?
Frequency	How often would an individual or group experience an impact to their well-being from the technology?

Next Steps

Use the Harms Model you developed to guide your product development work:

- Seek more information from stakeholders that you identified as potentially experiencing harm.
- Develop and validate hypothesis for addressing the areas you identified as having the highest potential for harm.
- Integrate the insights into your decisions throughout the technology development process: data collection and model training, system architecture, user experience design, product documentation, feedback loops, and communication capabilities and limitations of the technology.
- Explore [Community Jury](#).
- Assess and mitigate unfairness using Azure Machine Learning and the open-source [FairLearn package](#).

Other Responsible AI tools:

- [Responsible AI resource center](#)
- [Guidelines for Human AI interaction](#)
- [Conversational AI guidelines](#)

- [Inclusive Design guidelines](#)
- [AI Fairness checklist](#)

Additional references:

- [Downloadable booklet for assessing harms](#)
- [Value Sensitive Design](#)

Community jury

3/10/2022 • 7 minutes to read • [Edit Online](#)

Community jury, an adaptation of the [citizen jury](#), is a technique where diverse stakeholders impacted by technology are provided an opportunity to learn about a project, deliberate together, and give feedback on use cases and product design. This technique allows project teams to understand the perceptions and concerns of impacted stakeholders for effective collaboration.

A community jury is different from a focus group or market research; it allows the impacted stakeholders to hear directly from the subject matter experts in the product team, and cocreate collaborative solutions to challenging problems with them.

Benefits

This section discusses some important benefits of community jury.

Expert testimony

Members of a community jury hear details about the technology under consideration, directly from experts on the product team. These experts help highlight the capabilities in particular applications.

Proximity

A community jury allows decision-makers to hear directly from the community, and to learn about their values, concerns, and ideas regarding a particular issue or problem. It also provides a valuable opportunity to better understand the reasons for their conclusions.

Consensus

By bringing impacted individuals together and providing collaborative solutions and an opportunity for them to learn and discuss key aspects of the technology, a community jury can identify areas of agreement and build common ground solutions to challenging problems.

Community jury contributors

Product team

This group comprises owners who will bring the product to market, representative project managers, engineers, designers, data scientists, and others involved in the making of a product. Additionally, subject matter experts are included who can answer in-depth questions about the product.

Preparation

Effective deliberation and cocreation require ready-to-use answers to technical questions. As product owners, it is important to ensure technical details are collected prior to the start of the jury session.

Relevant artifacts could include:

- Documentation of existing protections, planned or in place.
- Data Flows, for example, what data types collected, who will have access, for how long, with what protections, and so on.
- Mockups or prototypes of proposed stakeholder collaborative solutions.

During the session

Along with providing an accessible presentation of technical details, product team witnesses should describe certain applications of the technology.

Relevant artifacts could include:

- Customer benefits
- Harms assessment and socio-technical impact
- Storyboards
- Competitive solutions and analyses
- Academic or popular media reports

Moderator

Bring on a neutral user researcher to ensure everyone is heard, avoiding domination of the communications by any one member. The moderator will facilitate brainstorms and deliberations, as well as educate jury members in uncovering bias, and ways to ask difficult questions. If a user researcher is not available, choose a moderator who is skilled at facilitating group discussions. Following the session, the moderator is responsible for the following:

- ensure that the agreed-upon compensation is provided to the jury members;
- produce a report that describes key insights, concerns, and recommendations, and
- share key insights and next steps with the jury, and thank them for their participation.

Preparation

- Structure the sessions so that there is ample time for learning, deliberation, and cocreation. This could mean having multiple sessions that go in-depth on different topics or having longer sessions.
- Pilot the jury with a smaller sample of community members to work out the procedural and content issues prior to the actual sessions.

During the session

- Ensure that all perspectives are heard, including those of the project team and those of the jury members. This minimizes group-thinking as well as one or two individuals dominating the discussion.
- Reinforce the value of the juror participation by communicating the plan for integrating jury feedback into the product planning process. Ensure that the project team is prepared to hear criticisms from the jury.

Jury members

These are direct and indirect stakeholders impacted by the technology, representative of the diverse community in which the technology will be deployed.

Sample size

A jury should be large enough to represent the diversity and collective will of the community, but not so large that the voices of quieter jurors are drowned out. It is ideal to get feedback from at least 16-20 individuals total who meet the criteria below. You can split the community jury over multiple sessions so that no more than 8-10 individuals are part of one session.

Recruitment criteria

Stratify a randomly selected participant pool so that the jury includes the demographic diversity of the community. Based on the project objectives, relevant recruitment criteria may include balancing across gender identity, age, [privacy index](#). Recruitment should follow good user research recruitment procedures and reach a wider community.

Session structure

Sessions typically last 2-3 hours. Add more or longer deep dive sessions, as needed, if aspects of the project require more learning, deliberation, and cocreation.

1. **Overview, introduction, and Q&A:** The moderator provides a session overview, then introduces the project team and explains the product's purpose, along with potential use cases, benefits, and harms. Questions are then accepted from community members. This session should be between 30 to 60

minutes long.

2. **Discussion of key themes:** Jury members ask in-depth questions about aspects of the project, fielded by the moderator. This session should also be between 30 to 60 minutes in length.
3. This step can be any one of the following options:
 - **Deliberation and cocreation:** This option is preferable. Jury members deliberate and co-create effective collaboration solutions with the project team. This is typically 30 to 60 minutes long.
 - **Individual anonymous survey:** Alternatively, conduct an anonymous survey of jury members. Anonymity may allow issues to be raised that would not otherwise be expressed. Use this 30-minute session if there are time constraints.
4. **Following the session:** The moderator produces a study report that describes key insights, concerns, and potential solutions to the concerns.

If the values of different stakeholders were in conflict with each other during the session and the value tensions were left unresolved, the product team would need to brainstorm solutions, and conduct a follow-up session with the jury to determine if the solutions adequately resolve their concerns.

Tips to run a successful, effective, and collaborative jury

- Ensure alignment of goals and outcomes with the project team before planning begins, including deliverables, recruitment strategy, and timeline. Consider including additional subject-matter experts relevant to the project, to address open questions/concerns.
- The consent to audio and video recording of the jury should follow your company's standard procedures for non-disclosure and consent that is obtained from participants during user research studies.
- Provide fair compensation to participants for the time they devote to participation. Whenever possible, participants should also be reimbursed for costs incurred as a result of study participation, for example, parking and transportation costs. Experienced user research recruiters can help determine fair gratuities for various participant profiles.
- Ensure that all perspectives are heard, minimizing group-thinking as well as one or two individuals dominating the discussion. This should include those of the project team and the jury members.
- Structure the sessions so that there is ample time for learning, deliberation, and cocreation. This could mean having multiple sessions going in-depth on different topics or having longer sessions.
- Pilot the jury with a smaller sample of community members to work out the procedural and content issues prior to the actual sessions.
- If the topic being discussed is related to personal data privacy, aim to balance the composition of the community jury to include individuals with different [privacy indices](#).
- Consider including session breaks and providing snacks/refreshments for sessions that are two hours or longer.
- Reinforce the value of the juror participation by communicating the plan for integrating jury feedback into the product planning process. Also ensure that the project team is prepared to hear criticisms from the jury.
- After the jury, in addition to publishing the research report, send out a thank you note to the volunteer participants of the jury, along with an executive summary of the key insights.

Additional information

Privacy index

The [Privacy Index](#) is an approximate measure for an individual's concern about personal data privacy, and is gauged using the following:

1. Consumers have lost all control over how personal information is collected and used by companies.
2. Most businesses handle the personal information they collect about consumers in a proper and confidential

way.

3. Existing laws and organizational practices provide a reasonable level of protection for consumer privacy today.

Participants are asked to provide responses to the above concerns using the scale of: 1 - Strongly Agree, 2 - Somewhat Agree, 3 - Somewhat Disagree, 4- Strongly Disagree, and classified into the categories below.

High/Fundamentalist => IF A = 1 or 2 AND B & C = 3 or 4

Low/unconcerned => IF A = 3 or 4 AND B & C = 1 or 2

Medium/pragmatist => All other responses

Next steps

Explore the following references for detailed information on this method:

- Jefferson center: creator of the Citizen's Jury method <https://jefferson-center.org/about-us/how-we-work/>
- Citizen's jury handbook http://www.rachel.org/files/document/Citizens_Jury_Handbook.pdf
- Case study: Connected Health Cities (UK)
 - [Project page](#)
 - [Final report](#)
 - [Jury specification](#)
 - [Juror's report](#)
- Case study: [Community Jury at Microsoft](#)

Architecture for startups

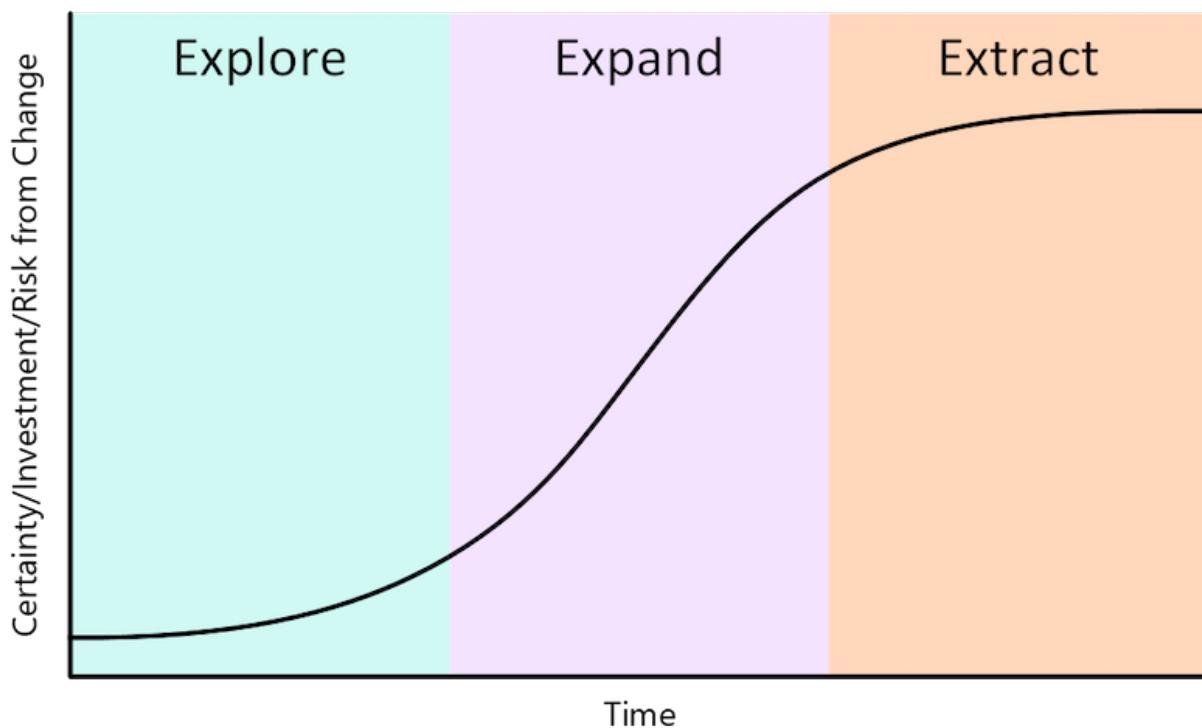
3/10/2022 • 5 minutes to read • [Edit Online](#)

Building a startup is a unique challenge. The core task is to find a place for an innovation as a product or service in the market. This process requires testing multiple assumptions that are built into the innovation. A successful startup must iterate through these assumptions, and grow and scale as its product gains product and market fit. After finding this fit, the startup must scale to meet market demands.

In the different startup life stages, developers, architects, and chief technical officers (CTOs) handle distinct phases of development. These stages require fundamentally different approaches and different technology choices. Part of the task is to establish which phase your startup is in. Choose the technologies and architectures that match that phase.

Innovation stages

Kent Beck describes a [three-stage process](#) of software product innovation. Those stages are *explore*, *expand*, and *extract*. You can think about the different parts of this process as a graph:



A graph showing a sigmoid curve plotted against a y-axis "Certainty/Investment/Risk of Change" and an x-axis "Time". The graph has three areas highlighted: the initial portion before upward inflection labeled "Explore", the high growth part of the sigmoid curve labeled "Expand" and the plateau labeled "Extract".

- The **Explore** stage starts with a low slope, where you're trying to find what works. Certainty is low, you only invest small amounts, and the risk from any changes you make is also low.
- At some point, the graph rises more rapidly. This rapid growth is the **Expand** stage. Your certainty greatly increases, you invest much more, and you're much more aware of risks.
- Finally, as the graph flattens out, you reach the **Extract** stage. The certainty, investment, and risk from change are all high, but the rate of growth has reached a plateau.

Explore

When your startup is in the exploration stage, your imperative is to invest small amounts of time and effort on many different product ideas. The fact that most ideas won't be right drives exploration. Only by iterating and learning can you find product and market fit. By making many small bets, you aim to find a product idea that pays off.

This stage requires discipline. It's easy to overinvest in an idea that you could test with less time and energy. A technologist finds it especially easy to fall into this trap. To make architectural choices that ease exploration, remember that you're exploring. You don't yet know if the current product idea is one that will scale.

From an architecture perspective, choose services that optimize for speed, cost, and options. Use managed services and platforms as a service (PaaS) like Azure App Service to get started quickly without worrying about complex infrastructure. Manage costs by choosing smaller instance sizes while you're exploring. Containers support developing with whatever tools make sense for you.

Build your first stack

As with your first product version, your first technology stack should be firmly rooted in exploration. That means the technology stack should ease rapid product iteration without wasting effort. You don't want to spend time or effort on infrastructure or architecture that isn't required for answering current questions.

During the exploration phase, you need to optimize for speed, cost, and optionality. Speed is about how fast you can build and move forward with an idea, or move onto the next idea. Cost is how much you're spending to run your infrastructure. Optionality describes how fast you can change directions given the current architecture.

It's important to balance cost, speed, and optionality. Too much focus on cost limits speed and optionality. Too much focus on speed can lead to increased costs and fewer options. Designing for too many options builds complexity, which increases costs and reduces speed.

Expand

Once your startup finds growth through exploration, you shift gears to expansion. You focus on removing any blockages to your product's and company's continued growth. From a technical perspective, you solve infrastructure scale challenges and increase development velocity. The goals are to meet your new customers' needs and advance your product roadmap.

Extend your architecture

As you iterate on your product, you'll inevitably find areas where your architecture needs extending. You might need to complete long-running tasks in the background, or handle frequent updates from internet-of-things (IoT) devices. You might need to add full-text search or artificial intelligence to your product.

You might need architectural changes to accommodate items on your roadmap. Resist the temptation to make those changes too far in advance. Extensions risk adding complexity to your architecture and infrastructure costs to your balance sheet.

In early startup stages, any architecture extension should be just-in-time. The extension should take only as much time and energy as needed to test the next hypothesis. Be ready to remove extensions to reduce complexity. Look for product features that your customers aren't using as opportunities to simplify your architecture and reduce your infrastructure spending.

Extract

In the extraction phase, the pace of growth slows as you reach the limits of the market opportunity. Because you expanded through the previous phase, there's now a lot to lose, so you take a more cautious approach. Margin expansion, cost reduction, and efficiency improvements characterize the extraction phase. During the extraction phase, be careful not to compromise the product for the customers you won in the expansion phase.

Handle growth and mature your stack

Once a product achieves product and market fit, many demands drive its architecture. Increased usage might require infrastructure scaling to handle load. New enterprise compliance requirements might require greater isolation. These changes are common steps in maturing a successful application.

Changes you make to handle growth and add maturity are different from extending architecture. These changes aren't functional requirements, but relate to unlocking scale. Increased scale can come from net new customers, increased usage from existing customers, and customers with higher regulatory requirements.

Resist the temptation to optimize prematurely. Make sure to take growth and maturation steps that can help you continue iterating and improving your product.

Next steps

- See and deploy an example [Core startup stack architecture](#).

Related resources

- [Best practices in cloud applications](#)
- [Ten design principles for Azure applications](#)

SaaS digital business journey on Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

Follow this guidance to get the basics right when you build your software as a service (SaaS) on Azure.

Evaluate Azure

Get informed and make the right platform and partnership decisions for your SaaS business.

- Evaluate these [Azure service architecture options](#) to find the right real-world Azure architecture for your service.
- Evaluate complementary [Azure Marketplace](#) services to enhance your app.
- Evaluate the [enablement options](#) that are available for Microsoft partners.
- Find which [migration options](#) meet your needs.

Get started with Azure

Get up and running quickly.

- Join the [Microsoft Partner Network \(MPN\)](#).
- Create a free [Azure Account](#).
- Set up [Azure Cost Management and Billing](#) to avoid surprise bills.
- Set up [Azure Advisor](#) to receive recommendations on cost optimization, security, performance, and more.
- Understand [Azure architectures](#) to find the right Azure architecture for your app.
- If you're designing for a startup, consider [architecture guidance for startups](#).
- Decide which tenant-isolation or sharing model to implement by reviewing [multitenant architectural guidance](#).

Build your app

Create a plan to reduce risk, whether you're developing an app or migrating one.

- Deploy the [enterprise-scale landing zone for small enterprise](#).
- [Implement source control, manage code changes](#), and [set up continuous integration/continuous delivery \(CI/CD\) pipelines](#).
- Complete your architecture in detail.
- Do a [Well Architected Framework Review](#).
- Design your [billing and metering strategy](#).

Deploy and publish your app

Ensure a successful deployment and launch to market.

- Deploy the production environment to your landing zone: see [Build your app](#).
- Perform a [security review](#) and review your [Azure secure score](#).
- Implement a customer metering strategy and create a [transactable marketplace offer](#).
- Launch your app.

Grow your audience

Accelerate your marketing strategy and innovate your product to drive competitive differentiation.

- Co-sell with microsoft [Microsoft co-sell status](#) to profit from the world-wide Microsoft partner ecosystem.
- [List your SaaS app with Microsoft Cloud Solution Provider partners](#) to expand your salesforce.
- [Extend your app to Microsoft Teams](#) in order to meet your users where they work all day long.
- [Integrate with Microsoft Power Platform](#) to empower your customers to build low-code and no-code apps that address their needs.
- Extend your app with [Azure Machine Learning](#) and [Azure Cognitive Services](#) to discover new insights and to respond to events programmatically.
- Schedule quarterly [Azure Advisor](#) reviews of cost-saving, performance, and security recommendations.
- Schedule an annual [cost optimization and well architected framework review](#).
- Stay up to date with the latest expert guidance from the [Microsoft SaaS Academy](#).

Next steps

- [Resources for architects and developers of multitenant solutions](#)
- [Multi-tenant SaaS database tenancy patterns](#)
- [Introduction to a multitenant SaaS app that uses the database-per-tenant pattern with Azure SQL Database](#)
- [Microsoft Learn for Azure](#)

Related resources

- [Architecture for startups](#)
- [Core startup stack architecture](#)
- [Architecting multitenant solutions on Azure](#)

Architect multitenant solutions on Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

A multitenant solution is one used by multiple customers, or *tenants*. Tenants are distinct from users. Multiple users from a single organization, company, or group form a single tenant. Examples of multitenant applications include:

- Business-to-business (B2B) solutions, such as accounting software, work tracking, and other software as a service (SaaS) products.
- Business-to-consumer (B2C) solutions, such as music streaming, photo sharing, and social network services.
- Enterprise-wide platform solutions, such as a shared Kubernetes cluster that's used by multiple business units within an organization.

When you build your own multitenant solution in Azure, there are several elements you need to consider that factor into your architecture.

In this series, we provide guidance about how to design, build, and operate your own multitenant solutions in Azure.

NOTE

In this series, we use the term *tenant* to refer to **your** tenants, which might be your customers or groups of users. Our guidance is intended to help you to build your own multitenant software solutions on top of the Azure platform.

Azure Active Directory (Azure AD) also includes the concept of a tenant to refer to individual directories, and it uses the term *multitenancy* to refer to interactions between multiple Azure AD tenants. Although the terms are the same, the concepts are not. When we need to refer to the Azure AD concept of a tenant, we disambiguate it by using the full term *Azure AD tenant*.

Scope

While Azure is itself a multitenant service, and some of our guidance is based on our experience with running large multitenant solutions, the focus of this series is on helping you build your own multitenant services, while harnessing the power of the Azure platform.

Additionally, when you design a solution, there are many areas you need to consider. The content in this section is specific to how you design for multitenancy. We don't cover all of the features of the Azure services, or all of the architectural design considerations for every application. You should read this guide in conjunction with the [Microsoft Azure Well-Architected Framework](#) and the documentation for each Azure service that you use.

Intended audience

The guidance provided in this series is applicable to anyone building a multitenant application in Azure. The audience also includes independent software vendors (ISVs) and startups who are building SaaS products, whether they are targeted for businesses or consumers. It also includes anyone building a product or platform that's intended to be used by multiple customers or tenants.

The content throughout this series is designed to be useful for technical decision-makers, like chief technology officers (CTOs) and architects, and anyone designing or implementing a multitenant solution on Microsoft Azure.

NOTE

Managed service providers (MSPs) manage and operate Azure environments on behalf of their customers, and work with multiple Azure Active Directory tenants in the process. This is another form of multitenancy, but it's focused on managing Azure resources across multiple Azure Active Directory tenants. This series isn't intended to provide guidance on these matters.

However, the series is likely to be helpful for ISVs who build software for MSPs, or for anyone else who builds and deploys multitenant software.

What's in this series?

The content in this series is composed of three main sections:

- **Architectural considerations for a multitenant solution:** This section provides an overview of the key requirements and considerations you need to be aware of when planning and designing a multitenant solution.

The pages in this section are particularly relevant for technical decision-makers, like chief technology officers (CTOs) and architects. Product managers will also find it valuable to understand how multitenancy affects their solutions. Additionally, anyone who works with multitenant architectures should have some familiarity with these principles and tradeoffs.

- **Architectural approaches for multitenancy:** This section describes the approaches you can consider when designing and building multitenant solutions, by using key cloud resource types. The section includes a discussion how to build multitenant solutions with compute, networking, storage, and data components, as well as deployment, configuration, governance, and cost management.

The architectural approaches are intended to be useful for solution architects and lead developers.

- **Service-specific guidance for a multitenant solution:** This section provides targeted guidance for specific Azure services. It includes discussions of the tenancy isolation models that you might consider for the components in your solution, as well as any features that are particularly relevant for a multitenant solution.

The service-specific guidance is useful for architects, lead developers, and anyone building or implementing Azure components for a multitenant solution.

Additionally, we provide a [list of related resources and links](#) for architects and developers of multitenant solutions.

Next steps

Review the [architectural considerations for a multitenant solution](#).

Architectural considerations for a multitenant solution

3/10/2022 • 2 minutes to read • [Edit Online](#)

When you're considering a multitenant architecture, there are several decisions you need to make and elements you need to consider.

In a multitenant architecture, you share some or all of your resources between tenants. This process means that a multitenant architecture can give you cost and operational efficiency. However, multitenancy introduces complexities, including the following:

- How do you define what a *tenant* is, for your specific solution? Does a tenant correspond to a customer, a user, or a group of users (like a team)?
- How will you deploy your infrastructure to support multitenancy, and how much isolation will you have between tenants?
- What commercial pricing models will your solution offer, and how will your pricing models affect your multitenancy requirements?
- What level of service do you need to provide to your tenants? Consider performance, resiliency, security, and compliance requirements, like data residency.
- How do you plan to grow your business or solution, and will it scale to the number of tenants you expect?
- Do any of your tenants have unusual or special requirements? For example, does your biggest customer need higher performance or stronger guarantees than others?
- How will you monitor, manage, automate, scale, and govern your Azure environment, and how will multitenancy impact this?

Whatever your architecture, it's essential that you have a clear understanding of your customers' or tenants' requirements. If you have made sales commitments to customers, or if you have contractual obligations or compliance requirements to meet, then you need to know what those requirements are, when you architect your solution. But equally, your customers may have implicit expectations about how things *should* work, or how you *should* behave, which could affect the way you design a multitenant solution.

As an example, imagine you're building a multitenant solution that you sell to businesses in the financial services industry. Your customers have very strict security requirements, and they need you to provide a comprehensive list of every domain name that your solution uses, so they can add it to their firewall's allowlist. This requirement affects the Azure services you use and the level of isolation that you have to provide between your tenants. They also require that their solution has a minimum level of resiliency. There may be many similar expectations, both explicit and implicit, that you need to consider across your whole solution.

In this section, we outline the considerations that you should give, the requirements you should elicit, and some of the tradeoffs you need to make, when you are planning a multitenant architecture.

Intended audience

The pages in this section are particularly relevant for technical decision-makers, like chief technology officers (CTOs) and architects, as well as product managers. The audience also includes independent software vendors (ISVs) and startups who develop SaaS solutions. Additionally, anyone who works with multitenant architectures should have some familiarity with these principles and tradeoffs.

Next steps

Consider different [tenancy models](#) for your solution.

Tenancy models to consider for a multitenant solution

3/10/2022 • 13 minutes to read • [Edit Online](#)

There are many different ways that you can design a solution to be multitenanted. Mostly this decision hinges on whether and how you share resources between your tenants. Intuitively, you might want to avoid sharing *any* resources, but this quickly becomes expensive, as your business scales and as you onboard more and more tenants.

It's helpful to think about the different models of multitenancy, by first understanding how you define tenants for your specific organization, what business drivers you have, and how you plan to scale your solution. On this page, we provide guidance for technical decision-makers about the tenancy models you can consider and their tradeoffs.

Define a tenant

First, you need to define a *tenant* for your organization. Consider who your customer is. In other words, who are you providing your services to? There are two common models:

- **Business-to-business (B2B).** If your customers are other organizations, you are likely to consider your tenants to be those customers. However, consider whether your customers might have divisions (teams or departments), or if they have a presence in multiple countries. You may need to consider having a single customer map to multiple tenants, if there are different requirements for these subgroups. Similarly, a customer might want to maintain two instances of your service, so they can keep their development and production environments separated from each other. Generally, a single tenant will have multiple users. For example, all of your customer's employees will be users within the same tenant.
- **Business-to-consumer (B2C).** If your customers are consumers, it's often more complicated to relate customers, tenants, and users. In some scenarios, each consumer could be their own tenant. However, consider whether your solution might be used by families, groups of friends, clubs, associations, or other groupings that might need to access and manage their data together. For example, a music-streaming service might support both individual users and families, and it might treat each of these account types differently, when it comes to separating them into tenants.

Your definition of *tenant* will impact some of the things that you need to consider or emphasize, when you architect your solution. For example, consider these different types of tenants:

- If your tenants are individual people or families, you may need to be particularly concerned about how you handle personal data, and the data sovereignty laws within each jurisdiction you serve.
- If your tenants are businesses, you may need to be mindful of your customers' requirements for regulatory compliance, the isolation of their data, and ensuring you meet a specified service-level objective (SLO), such as uptime or service availability.

How do you decide which model to use?

Selecting a tenancy model isn't just a technical decision; it's also a commercial decision that you need to make. You need to consider questions like the following:

- What are your business objectives?
- Will your customers accept all forms of multitenancy? How would each multitenancy model impact your compliance requirements, or your customer's compliance requirements?

- Will a single-tenant solution scale to your future growth aspirations?
- How large is your operations team, and how much of your infrastructure management are you able to automate?
- Do your customers expect you to meet service-level agreements (SLAs), or do you have SLOs that you are aiming for?

If you expect that your business is going to scale to a large number of customers, it will be very important to deploy shared infrastructure. Otherwise, you'll have to maintain a large and ever-growing fleet of instances. Deploying individual Azure resources, for each customer, is likely to be unsustainable, unless you provision and use a dedicated subscription, per tenant. When sharing the same Azure subscription across multiple tenants, [Azure resource quotas and limits](#) might start to apply, and the operational costs to deploy and reconfigure these resources become higher, with each new customer.

Conversely, if you expect that your business is only going to have a few customers, it might be worth considering to have single-tenant resources that are dedicated to each customer. Similarly, if your customers' isolation requirements are high, a single-tenant infrastructure might be appropriate.

Logical and physical tenants

Next, you need to determine what a tenant means for your particular solution, and whether you should distinguish between *logical* and *physical* tenants.

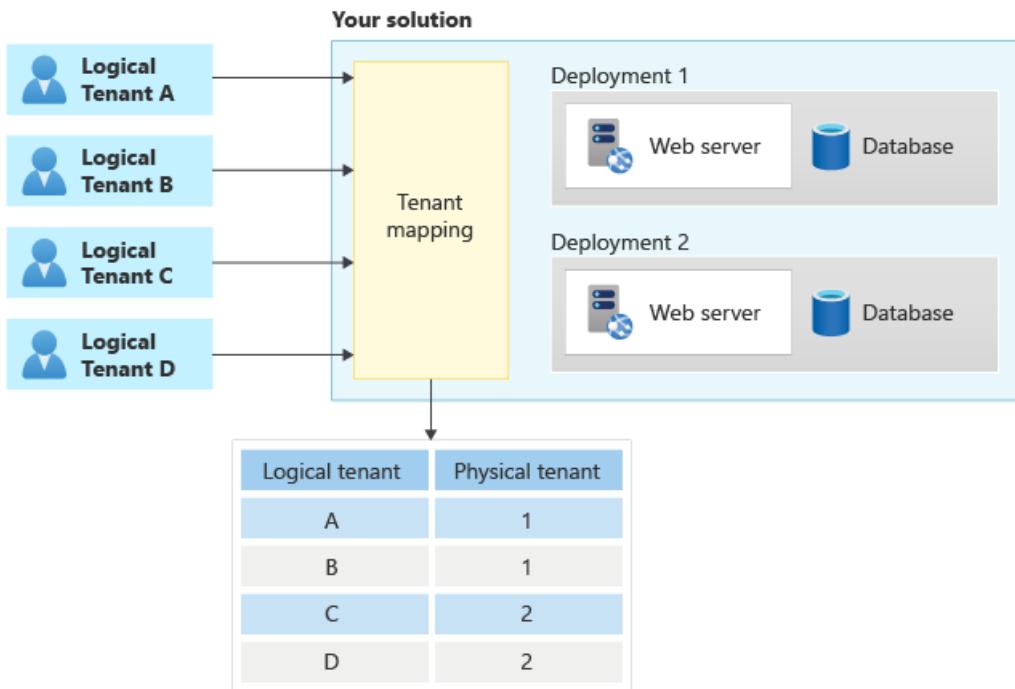
For example, consider a music streaming service. Initially, you might build a solution that can easily cope with thousands (or even tens of thousands) of users. As you continue to grow, though, you might find that you need to duplicate your solution or some of its components, in order to scale to your new customer demand. This means that you'll need to work out how to assign specific customers to specific instances of your solution. You might do this randomly, or geographically, or by filling up a single instance and then spilling over to another. However, you will probably need to maintain a record of which customers you have and which infrastructure their data and applications are available on, so that you can route their traffic to the correct infrastructure. In this example, you might represent each user as a separate logical tenant, and then map the users to physical tenants that represent the different instances you've deployed. You have a one-to-many mapping between logical and physical tenants, and you can move logical tenants between physical tenants at your own discretion.

In contrast, consider a company that builds cloud software for legal firms. Your customers might insist on their own dedicated infrastructure to maintain their compliance standards. Therefore, you need to be prepared to deploy and manage many different instances of your solution, right from the start. In this example, a logical and physical tenant is the same thing.

A key difference between logical and physical tenants is how isolation is enforced. When multiple logical tenants share a single set of infrastructure, you typically rely on your application code and a tenant identifier in a database, to keep each tenant's data separate. When you have physical tenants, they have their own infrastructure, and so it may be less important for your code to be aware that it's operating in a multitenant environment.

Sometimes, you'll see physical tenants referred to as *deployments*, *supertenants*, or *stamps*. In the rest of this document, we use the terms *deployments* and *physical deployments*, to avoid confusion between logical and physical tenants.

When you receive a request for a specific logical tenant, you need to map it to the physical deployment that holds that tenant's data, as illustrated below:

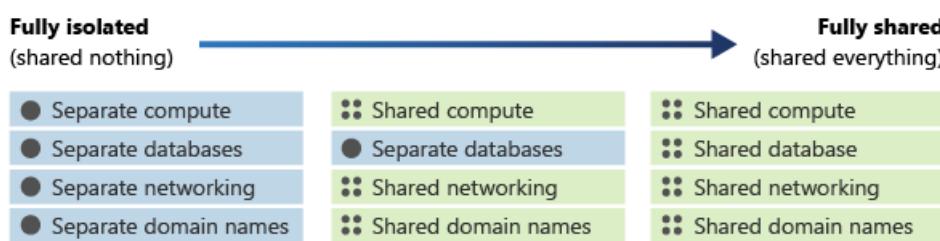


Tenant isolation

One of the biggest considerations when designing a multitenant architecture is the level of isolation that each tenant needs. Isolation can mean different things:

- Having a single set of shared infrastructure, with separate instances of your application and separate databases for each tenant.
- Sharing some common resources, while keeping other resources separate for each tenant.
- Keeping data on a separate physical infrastructure. In the cloud, this might require separate Azure resources for each tenant, or it could even mean literally deploying a separate physical infrastructure, by using [dedicated hosts](#).

Rather than thinking of isolation as being a discrete property, you should think about isolation as being a continuum. You can deploy components of your architecture that are more or less isolated than other components in the same architecture, depending on your requirements. The following diagram demonstrates a continuum of isolation:



The level of isolation impacts many aspects of your architecture, including the following:

- **Security.** If you share infrastructure between multiple tenants, you need to be especially careful not to access data from one tenant when returning responses to another. You need a strong foundation for your identity strategy, and you need to consider both tenant and user identity within your authorization process.
- **Cost.** Shared infrastructure can be used by multiple tenants, so it's cheaper.
- **Performance.** If you're sharing infrastructure, your system's performance may suffer as more customers use it, since the resources may be consumed faster.
- **Reliability.** If you're using a single set of shared infrastructure, a problem with one tenant's components can result in an outage for everyone.

- **Responsiveness to individual tenants' needs.** When you deploy infrastructure that is dedicated to one tenant, you may be able to tune the configuration for the resources for that specific tenant's requirements. You might even consider this in your pricing model, where you enable customers to pay more for isolated deployments.

Your solution architecture can influence the options that you've got available to you for isolation. For example, let's think about an example three-tier solution architecture:

- Your user interface tier might be a shared multitenant web app, and all of your tenants access a single hostname.
- Your middle tier could be a shared application layer, with shared message queues.
- Your data tier could be isolated databases, tables, or blob containers.

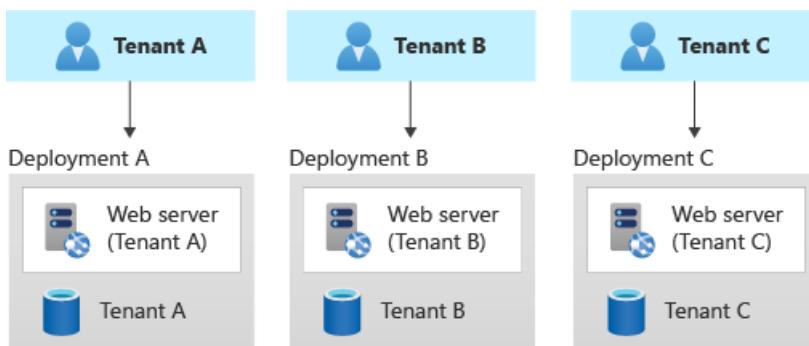
You can consider mixing and matching different levels of isolation at each tier. Your decision about what is shared and what is isolated will be based on many considerations, including cost, complexity, your customers' requirements, and the number of resources that you can deploy before reaching Azure quotas and limits.

Common tenancy models

Once you've established your requirements, evaluate them against some common tenancy models and patterns.

Automated single-tenant deployments

In an automated single-tenant deployment model, you deploy a dedicated set of infrastructure for each tenant, as illustrated in this example:



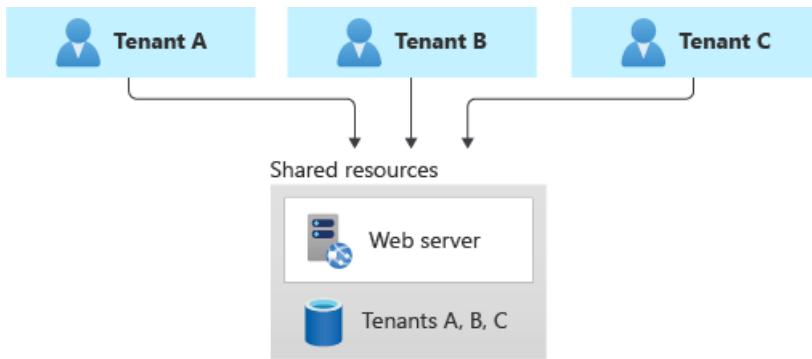
Your application is responsible for initiating and coordinating the deployment of each tenant's resources. Typically, solutions built using this model make extensive use of infrastructure as code (IaC) or the Azure Resource Manager APIs. You might use this approach when you need to provision entirely separate infrastructures for each of your customers. Consider the [Deployment Stamps pattern](#) when planning your deployment.

Benefits: A key benefit of this approach is that data for each tenant is isolated, which reduces the risk of accidental leakage. This can be important to some customers with high regulatory compliance overhead. Additionally, tenants are unlikely to affect each other's system performance, which is sometimes called the *noisy neighbor* problem. Updates and changes can be rolled out progressively across tenants, which reduces the likelihood of a system-wide outage.

Risks: Your cost efficiency is low, because you aren't sharing infrastructure between your tenants. If a single tenant requires spending a certain amount on infrastructure, then it's likely that 100 tenants will require 100 times that cost, in expenditure. Additionally, ongoing maintenance (like applying new configuration or software updates) is likely to be time-consuming. Consider automating your operational processes, and consider applying changes progressively through your environments. You should also consider other cross-deployment operations, like reporting and analytics across your whole estate. Likewise, ensure you plan for how you can query and manipulate data across multiple deployments.

Fully multitenant deployments

At the opposite extreme, you can consider a fully multitenant deployment, where all components are shared. You only have one set of infrastructure to deploy and maintain, and all tenants use it, as illustrated in the following diagram:



Benefits: This model is attractive because of the lower cost to operate a solution with shared components. Even if you need to deploy higher tiers or SKUs of resources, it's still often the case that the overall deployment cost is lower than a set of single-tenant resources. Additionally, if a user or tenant needs to move their data into another logical tenant, you don't have to migrate data between two separate deployments.

Risks:

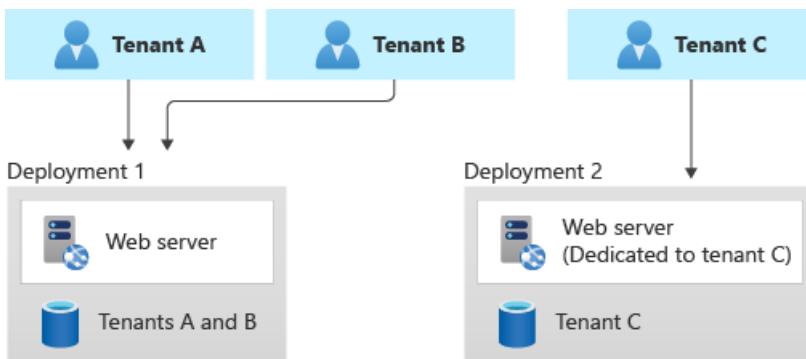
- Take care to ensure you separate data for each tenant, and do not leak data between tenants. You may need to manage sharding your data yourself. Additionally, you may need to be concerned about the effects that individual tenants can have on the overall system. For example, if a single large tenant tries to perform a heavy query or operation, will it affect other tenants?
- Determine how you [track and associate your Azure costs to tenants](#), if this is important to you. Maintenance can be simpler with a single deployment, since you only have to update one set of resources. However, it's also often riskier, since any changes may affect your entire customer base.
- Scale can be a factor to consider as well. You are more likely to reach [Azure resource scale limits](#) when you have a shared set of infrastructure. For example, if you use a storage account as part of your solution, then as your scale increases, the number of requests to that storage account could reach the limit of what the storage account can handle. To avoid hitting a resource quota limit, you might consider deploying multiple instances of your resources (for example, multiple AKS clusters or storage accounts), or you might even consider distributing your tenants across resources that you've deployed into multiple Azure subscriptions.
- There is likely to be a limit to how far you can scale a single deployment, and the costs of doing so may increase non-linearly. For example, if you have a single, shared database, when you run at very high scale you may exhaust its throughput and have to pay increasingly more for increased throughput, to keep up with your demand.

Vertically partitioned deployments

You don't have to sit at the extremes of these scales. Instead, you could consider vertically partitioning your tenants, with the following steps:

- Use a combination of single-tenant and multitenant deployments. For example, you might have most of your customers' data and application tiers on multitenant infrastructures, but you might deploy single-tenant infrastructures for customers who require higher performance or data isolation.
- Deploy multiple instances of your solution geographically, and have each tenant pinned to a specific deployment. This is particularly effective when you have tenants in different geographies.

Here's an example that illustrates a shared deployment for some tenants, and a single-tenant deployment for another:

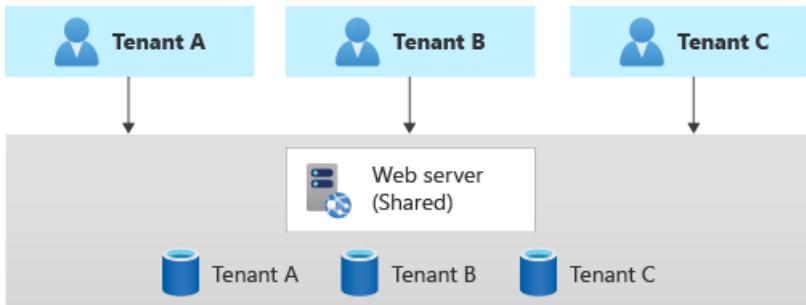


Benefits: Since you are still sharing infrastructure, you can still gain some of the cost benefits of having shared multitenant deployments. You can deploy cheaper, shared resources for certain customers, like those who are trying your service with a trial. You can even bill customers a higher rate to be on a single-tenant deployment, thereby recouping some of your costs.

Risks: Your codebase will likely need to be designed to support both multitenant and single-tenant deployments. If you plan to allow migration between infrastructures, you need to consider how you migrate customers from a multitenant deployment to their own single-tenant deployment. You also need to have a clear understanding of which of your logical tenants are on which sets of physical infrastructure, so that you can communicate information about system issues or upgrades to the relevant customers.

Horizontally partitioned deployments

You can also consider horizontally partitioning your deployments. This means you have some shared components, while maintaining other components with single-tenant deployments. For example, you could build a single application tier, and then deploy individual databases for each tenant, as shown in this illustration:



Benefits: Horizontally partitioned deployments can help you mitigate a noisy-neighbor problem, if you've identified that most of the load on your system is due to specific components that you can deploy separately for each tenant. For example, your databases might absorb most of your system's load, because the query load is high. If a single tenant sends a large number of requests to your solution, the performance of a database might be negatively affected, but other tenants' databases (and shared components, like the application tier) remain unaffected.

Risks: With a horizontally partitioned deployment, you still need to consider the automated deployment and management of your components, especially the components used by a single tenant.

Test your isolation model

Whichever isolation model you select, ensure you test your solution to verify that one tenant's data isn't accidentally leaked to another and that any [noisy neighbor](#) effects are acceptable. Consider using [Azure Chaos Studio](#) to deliberately introduce faults that simulate real-world outages and verify the resiliency of your solution even when components are malfunctioning.

Next steps

Consider the [lifecycle of your tenants](#).

Tenant lifecycle considerations in a multitenant solution

3/10/2022 • 4 minutes to read • [Edit Online](#)

When you're considering a multitenant architecture, it's important to consider all of the different stages in a tenant's lifecycle. On this page, we provide guidance for technical decision-makers about the stages of the lifecycle and the important considerations for each stage.

Trial tenants

For SaaS solutions, consider that many customers request or require trials, before they commit to purchase a solution. Trials bring along the following unique considerations:

- Should the trial data be subject to the same data security, performance, and service-level requirements as the data for full customers?
- Should you use the same infrastructure for trial tenants as for full customers, or should you have dedicated infrastructure for trial tenants?
- If customers purchase your service after a trial, how will they migrate the data from their trial tenants into their paid tenants?
- Are there limits around who can request a trial? How can you prevent abuse of your solution?
- What limits do you want or need to place on trial customers, such as time limits, feature restrictions, or limitations around performance?

Onboard new tenants

When onboarding a new tenant, consider the following:

- Will onboarding be a self-service, automated, or manual process?
- Does the customer have any specific requirements for data residency? For example, are there data sovereignty regulations in effect?
- Does the customer have to meet any compliance standards (such as PCI DSS, HIPAA, and so on)?
- Does the customer have any specific disaster recovery requirements, such as a recovery time objective (RTO) or a recovery point objective (RPO)? Are these different from the guarantees that you provide to other customers?
- What information do you require, to be able to fully onboard the customer?
- Does the platform provide different pricing options and billing models?
- Does the customer require pre-production environments? And are there set expectations on availability for that environment? Is it transient (on-demand) or always available?

Once tenants have been onboarded, they move into a 'business as usual' mode. However, there are still several important lifecycle events that can occur, even when they are in this mode.

Update tenants' infrastructure

You will need to consider how you apply updates to your tenants' infrastructures. Different tenants may have updates applied at different times. See [Updates](#) for other considerations about updating tenants' deployments.

Scale tenants' infrastructure

Consider whether your tenants might have seasonal business patterns, or otherwise change the level of consumption for your solution. For example, if you provide a solution to retailers, you expect that certain times of the year will be particularly busy in some geographic regions, and quiet at other times. Consider whether this affects the way you design and scale your solution, and be aware of *noisy neighbor* issues, when a subset of tenants scales unexpectedly and impacts the performance to other tenants. You can consider applying mitigations, which might include scaling individual tenants' infrastructures, moving tenants between deployments, and provisioning a sufficient level of capacity to handle spikes and troughs in traffic.

Move tenants between infrastructures

You might need to move tenants between infrastructures for a number of reasons, including the following:

- You are vertically partitioning your customers and choose to rebalance your tenants across your infrastructures or deployments.
- Customers are upgrading a SKU or pricing tier, and they need to be moved to a single-tenant, dedicated deployment with higher isolation from other tenants.
- Customers request their data to be moved to a dedicated data store.
- Customers require their data be moved to a new geographic region. This might occur during company acquisitions, or when laws or geopolitical situations change.

Consider how you move your tenants' data, as well as redirect requests to the new set of infrastructure that hosts their instance. You should also consider whether moving a tenant will result in downtime, and make sure tenants are fully aware of this.

Merge and split tenants

It's tempting to think of tenants or customers as static, unchanging entities. However, in reality, this often isn't true. For example:

- In business scenarios, companies might be acquired or merge, including companies located in different geographic regions.
- Similarly, in business scenarios, companies might split or divest.
- In consumer scenarios, individual users might join or leave families.

Consider whether you need to provide capabilities to manage the merging and separation of data, user identities, and resources. Also, consider how data ownership affects your handling of merge and split operations. For example, consider a consumer photography application built for families to share photos with one another. Are the photos owned by the individual family members who contributed them, or by the family as a whole? If users leave the family, should their data be removed or remain in the family's data set? If users join another family, should their old photos move with them?

Offboard tenants

It's also inevitable that tenants will occasionally need to be removed from your solution. In a multitenant solution, this brings along some important considerations, including the following:

- How long should you maintain the customer data? Are there legal requirements to destroy data, after a certain period of time?
- Should you provide the ability for customers to be re-onboarded?
- If you run shared infrastructure, do you need to rebalance the allocation of tenants to infrastructure?

Deactivate and reactivate tenants

There are situations where a customer's account might need to be deactivated or reactivated. For example:

- The customer has requested deactivation. In a consumer system, a customer might opt to unsubscribe.
- The customer can't be billed, and you need to deactivate the subscription.

Deactivation is separate to offboarding in that it's intended to be a temporary state. However, after a period of time, you might choose to offboard a deactivated tenant.

Next steps

Consider the [pricing models](#) you will use for your solution.

Pricing models for a multitenant solution

3/10/2022 • 20 minutes to read • [Edit Online](#)

A good pricing model ensures that you remain profitable as the number of tenants grows and as you add new features. An important consideration when developing a commercial multitenant solution is how to design pricing models for your product. On this page, we provide guidance for technical decision-makers about the pricing models you can consider and the tradeoffs involved.

When you determine the pricing model for your product, you need to balance the *return on value* (ROV) for your customers with the *cost of goods sold* (COGS) to deliver the service. Offering more flexible commercial models (for a solution) might increase the ROV for customers, but it might also increase the architectural and commercial complexity of the solution (and therefore also increase your COGS).

Some important considerations that you should take into account, when developing pricing models for a solution, are as follows:

- Will the COGS be higher than the profit you earn from the solution?
- Can the COGS change over time, based on growth in users or changes in usage patterns?
- How difficult is it to [measure and record the information required to operate the pricing model](#)? For example, if you plan to bill your customers based on the number of API calls they make, have you identified how you'll measure the API calls made by each customer?
- Does your profitability depend on ensuring customers use your solution in a limited way?
- If a customer overuses the solution, does that mean you're no longer profitable?

There are some key factors that influence your profitability:

- **Azure service pricing models.** The pricing models of the Azure or third-party services that make up your solution may affect which models will be profitable.
- **Service usage patterns.** Users may only need to access your solution during their working hours or may only have a small percentage of high-volume users. Can you reduce your COGS by reducing the unused capacity when your usage is low?
- **Storage growth.** Most solutions accumulate data over time. More data means a higher cost to store and protect it, reducing your profitability per tenant. Can you set storage quotas or enforce a data retention period?
- **Tenant isolation.** The [tenancy model](#) you use affects the level of isolation you have between your tenants. If you share your resources, do you need to consider how tenants might over-utilize or abuse the service? How will this affect your COGS and performance for everyone? Some pricing models are not profitable without additional controls around resource allocation. For example, you might need to implement service throttling to make a flat-rate pricing model sustainable.
- **Tenant lifecycle.** For example, solutions with high customer churn rates, or services that require a greater on-boarding effort, may suffer lower profitability--especially if they are priced using a consumption-based model.
- **Service level requirements.** Tenants that require higher levels of service may mean your solution isn't profitable anymore. It's critical that you're clear about your customers' service-level expectations and any obligations you have, so that you can plan your pricing models accordingly.

Common pricing models

There are a number of common pricing models that are often used with multitenant solutions. Each of these pricing models has associated commercial benefits and risks, and requires additional architectural

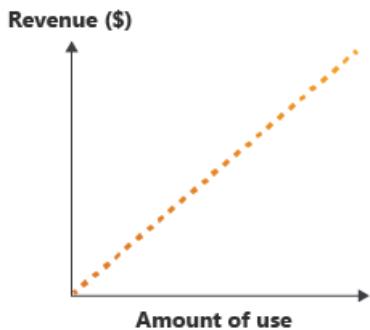
considerations. It's important to understand the differences between these pricing models, so that you can ensure your solution remains profitable as it evolves.

NOTE

You can offer multiple models for a solution or combine models together. For example, you could provide a per-user model for your customers that have fairly stable user numbers, and you can also offer a consumption model for customers who have fluctuating usage patterns.

Consumption-based pricing

A consumption model is sometimes referred to as *pay-as-you-go*, or *PAYG*. As the use of your service increases, your revenue increases:



When you measure consumption, you can consider simple factors, such as the amount of data being added to the solution. Alternatively, you might consider a combination of usage attributes together. Consumption models offer a number of benefits, but they can be difficult to implement in a multitenant solution.

Benefits: From your customers' perspective, there is minimal upfront investment that's required to use your solution, so that this model has a low barrier to entry. From your perspective as the service operator, your hosting and management costs increase as your customers' usage and your revenue increases. This increase can make it a highly scalable pricing model. Consumption pricing models work especially well when the Azure services that are used in the solution are consumption-based too.

Complexity and operational cost: Consumption models rely on accurate measurements of usage and on splitting this usage by tenant. This can be challenging, especially in a solution with many distributed components. You need to keep detailed consumption records for billing and auditing.

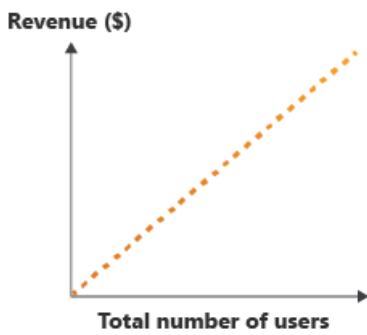
Risks: Consumption pricing can motivate your customers to reduce their usage of your system, in order to reduce their costs. Additionally, consumption models result in unpredictable revenue streams. You can mitigate this by offering *capacity reservations*, where customers pay for some level of consumption upfront. You, as the service provider, can use this revenue to invest in improvements in the solution, to reduce the operational cost or to increase the return on value by adding features.

NOTE

Implementing and supporting capacity reservations may increase the complexity of the billing processes within your application. You might also need to consider how customers get refunds or exchange their capacity reservations, and these processes can also add commercial and operational challenges.

Per-user pricing

A per-user pricing model involves charging your customers based on the number of people using your service, as demonstrated in the following diagram.



Per-user pricing models are very common, due to their simplicity to implement in a multitenant solution. However, they are associated with several commercial risks.

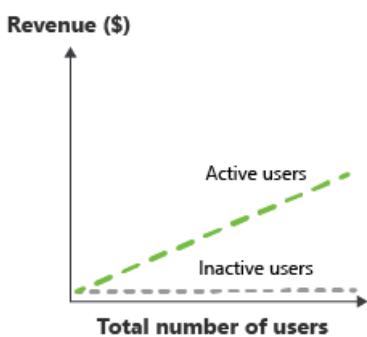
Benefits: When you bill your customers for each user, it's easy to calculate and forecast your revenue stream. Additionally, assuming that you have fairly consistent usage patterns for each user, then revenue increases at the same rate as service adoption, making this a scalable model.

Complexity and operational cost: Per-user models tend to be easy to implement. However, in some situations, you need to measure per-user consumption, which can help you to ensure that the COGS for a single user remains profitable. By measuring the consumption and associating it with a particular user, you can increase the operational complexity of your solution.

Risks: Different user consumption patterns might result in a reduced profitability. For example, heavy users of the solution might cost more to serve than light users. Additionally, the actual return on value (ROV) for the solution is not reflected by the actual number of user licenses purchased.

Per-active user pricing

This model is similar to [per-user pricing](#), but rather than requiring an upfront commitment from the customer on the number of expected users, the customer is only charged for users that actually log into and use the solution over a period (as shown in the following diagram).



You can measure this in whatever period makes sense. Monthly periods are common, and then this metric is often recorded as *monthly active users* or *MAU*.

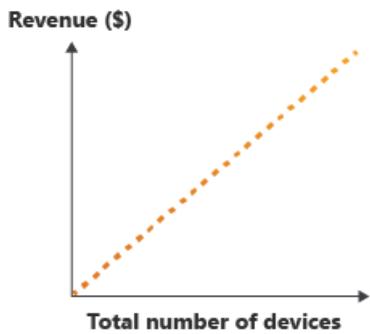
Benefits: From your customers' perspective, this model requires a low investment and risk, because there is minimal waste; unused licenses aren't billable. This makes it particularly attractive when marketing the solution or growing the solution for larger enterprise customers. From your perspective as the service owner, your ROV is more accurately reflected to the customer by the number of monthly active users.

Complexity and operational cost: Per-active user models require you to record actual usage, and to make it available to a customer as part of the bill. Measuring per-user consumption helps to ensure profitability is maintained with the COGS for a single user, but again it requires additional work to measure the consumption for each user.

Risks: Like per-user pricing, there is a risk that the different consumption patterns of individual users may affect your profitability. Compared to per-user pricing, per-active user models have a less predictable revenue stream. Additionally, [discount pricing](#) doesn't provide a useful way of stimulating growth.

Per-unit pricing

In many systems, the number of users isn't the element that has the greatest effect on the overall COGS. For example, in device-oriented solutions, also referred to as the *internet of things* or *IoT*, the number of devices often has the greatest impact on COGS. In these systems, a per-unit pricing model can be used, where you define what a *unit* is, such as a device. See the following diagram.



Additionally, some solutions have highly variable usage patterns, where a small number of users has a disproportionate impact on the COGS. For example, in a solution sold to brick-and-mortar retailers, a per-store pricing model might be appropriate.

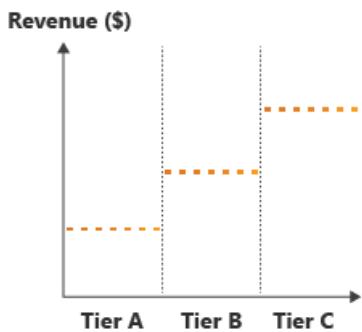
Benefits: In systems where individual users don't have a significant effect on COGS, per-unit pricing is a better way to represent the reality of how the system scales and the resulting impact to COGS. It also can improve the alignment to the actual patterns of usage for a customer. For many IoT solutions, where each device generates a predictable and constant amount of consumption, this can be an effective model to scale your solution's growth.

Complexity and operational cost: Generally, per-unit pricing is easy to implement and has a fairly low operational cost. However, the operational cost can become higher if you need to differentiate and track usage by individual units, such as devices or retail stores. Measuring per-unit consumption helps you ensure your profitability is maintained, since you can determine the COGS for a single unit.

Risks: The risks of a per-unit pricing model are similar to per-user pricing. Different consumption patterns by some units may mean that you have reduced profitability, such as if some devices or stores are much heavier users of the solution than others.

Feature- and service-level based pricing

You may choose to offer your solution with different tiers of functionality at different price points. For example, you might provide two monthly flat-rate or per-unit prices, one being a basic offering with a subset of features available, and the other presenting the comprehensive set of your solution's features. See the following diagram.



This model may also offer different service-level agreements for different tiers. For example, your basic tier may offer 99.9% uptime, whereas a premium tier may offer 99.99%. The higher service-level agreement (SLA) could be implemented by using services and features that enable higher [availability targets](#).

Although this model can be commercially beneficial, it does require mature engineering practices to do well. With careful consideration, this model can be very effective.

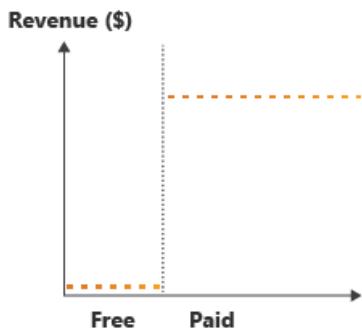
Benefits: Feature-based pricing is often attractive to customers, since they can select a tier based on the feature set or service level they need. It also provides you with a clear path to upsell your customers with new features or higher resiliency for those who require it.

Complexity and operational cost: Feature-based pricing models can be complex to implement, since they require your solution to be aware of the features that are available at each price tier. Feature toggles can be an effective way to provide access to certain subsets of functionality, but this requires ongoing maintenance. Also, toggles increase the overhead to ensure high quality, because there will be more code paths to test. Enabling higher service availability targets in some tiers may require additional architectural complexity, to ensure the right set of infrastructure is used for each tier, and this process may increase the operational cost of the solution.

Risks: Feature-based pricing models can become complicated and confusing, if there are too many tiers or options. Additionally, the overhead involved in dynamically toggling features can slow down the rate at which you deliver additional features.

Freemium pricing

You might choose to offer a free tier of your service, with basic functionality and no service-level guarantees. You then might offer a separate paid tier, with additional features and a formal service-level agreement (as shown in the following diagram).



The free tier may also be offered as a time-limited trial, and during the trial your customers might have full or limited functionality available. This is referred to as a freemium model, which is effectively an extension of the [feature-based pricing model](#).

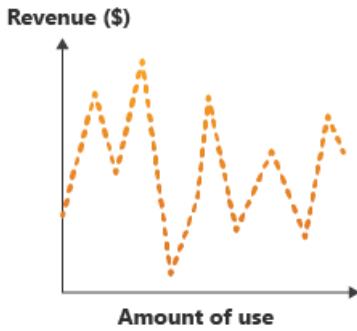
Benefits: It's very easy to market a solution when it's free.

Complexity and operational cost: All of the complexity and operational cost concerns apply from the feature-based pricing model. However, you also have to consider the operational cost involved in managing free tenants. You might need to ensure that stale tenants are offboarded or removed, and you must have a clear retention policy, especially for free tenants. When promoting a tenant to a paid tier, you might need to move the tenant between Azure services, to obtain higher SLAs.

Risks: You need to ensure that you provide a high enough ROV for tenants to consider switching to a paid tier. Additionally, the cost of providing your solution to customers on the free tier needs to be covered by the profit margin from those who are on paid tiers.

Cost of goods sold pricing

You might choose to price your solution so that each tenant only pays the cost of operating their share of the Azure services, with no added profit margin. This model - also called *pass through cost or pricing* - is sometimes used for multitenant solutions that are not intended to be a profit center.



The cost of goods sold model is a good fit for internally facing multitenant solutions. Each organizational unit corresponds to a tenant, and the costs of your Azure resources need to be spread between them. It might also be appropriate where revenue is derived from sales of other products and services that consume or augment the multitenant solution.

Benefits: Because this model does not include any added margin for profit, the cost to tenants will be lower.

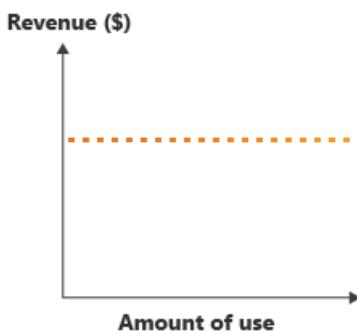
Complexity and operational cost: Similar to the consumption model, cost of goods sold pricing relies on [accurate measurements of usage](#) and on splitting this usage by tenant. Tracking consumption can be challenging, especially in a solution with many distributed components. You need to keep detailed consumption records for billing and auditing.

For internally facing multitenant solutions, tenants might accept approximate cost estimates and have more relaxed billing audit requirements. These relaxed requirements reduce the complexity and cost of operating your solution.

Risks: Cost of goods sold pricing can motivate your tenants to reduce their usage of your system, in order to reduce their costs. However, because this model is used for applications that are not profit centers, this might not be a concern.

Flat-rate pricing

In this model, you charge a flat rate to a tenant for access to your solution, for a given period of time. The same pricing applies regardless of how much they use the service, the number of users, the number of devices they connect, or any other metric. See the following diagram.



This is the simplest model to implement and for customers to understand, and it's often requested by enterprise customers. However, it can easily become unprofitable if you need to continue to add new features or if tenant consumption increases without any additional revenue.

Benefits: Flat-rate pricing is easy to sell, and it's easy for your customers to understand and budget for.

Complexity and operational cost: Flat-rate pricing models can be easy to implement because billing customers doesn't require any metering or tracking consumption. However, while not essential, it's advisable to measure per-tenant consumption to ensure that you're measuring COGS accurately and to ensure that your profitability is maintained.

Risks: If you have tenants who make heavy use of your solution, then it's easy for this pricing model to become

unprofitable.

Discount pricing

Once you've defined your pricing model, you might choose to implement commercial strategies to incentivize growth through discount pricing. *Discount pricing* can be used with consumption, per-user, and per-unit pricing models.

NOTE

Discount pricing doesn't typically require architectural considerations, beyond adding support for a more complex billing structure. A complete discussion into the commercial benefits of discounting is beyond the scope of this document.

Common discount pricing patterns include:

- **Fixed pricing.** You have the same cost for each user, unit, or amount of consumption, no matter how much is purchased or consumed. This is the simplest approach. However, customers who make heavy use of your solution may feel like they should benefit from economies of scale by getting a discount.
- **Degressive pricing.** As customers purchase or consume more units, you reduce the cost per unit. This is more commercially attractive to customers.
- **Step pricing.** You reduce the cost per unit, as customers purchase more. However, you do so in step changes, based on predefined ranges of quantity. For example, you might charge a higher price for the first 100 users, then a lower price for the 101st to 200th user, then a lower price again after that. This can be more profitable.

The following diagram illustrates these pricing patterns.



Non-production environment discounts

In many cases, customers require access to a non-production environment that they can use for testing, training, or for creating their own internal documentation. Non-production environments usually have lower consumption requirements and costs to operate. For example, non-production environments often aren't subject to service-level agreements (SLAs), and [rate limits](#) might be set at lower values. You might also consider more aggressive [autoscaling](#) on your Azure services.

Equally, customers often expect non-production environments to be significantly cheaper than their production environments. There are several alternatives that might be appropriate, when you provide non-production environments:

- Offer a [freemium tier](#), like you might already do for paid customers. This should be carefully monitored, as some organizations might create many testing and training environments, which will consume additional

resources to operate.

NOTE

Time-limited trials using freemium tiers aren't usually suitable for testing and training environments. Customers usually need their non-production environments to be available for the lifetime of the production service.

- Offer a testing or training tier of your service, with [lower usage limits](#). You may choose to restrict the availability of this tier to customers who have an existing paid tenant.
- Offer a discounted [per-user](#), [per-active user](#), or [per-unit](#) pricing for non-production tenants, with a lower or no service level agreement.
- For tenants using [flat-rate pricing](#), non-production environments might be negotiated as part of the agreement.

NOTE

Feature-based pricing is not usually a good option for non-production environments, unless the features offered are the same as what the production environment offers. This is because tenants will usually want to test and provide training on all the same features that are available to them in production.

Unprofitable pricing models

An unprofitable pricing model costs you more to deliver the service than the revenue you earn. For example, you might charge a flat rate per tenant without any limits for your service, but then you would build your service with consumption-based Azure resources and without per-tenant [usage limits](#). In this situation, you may be at risk of your tenants overusing your service and, thus, making it unprofitable to serve them.

Generally, you want to avoid unprofitable pricing models. However, there are situations where you might choose to adopt an unprofitable pricing model, including:

- A free service is being offered to enable growth.
- Additional revenue streams are provided by services or add-on features.
- Hosting a specific tenant provides another commercial value, such as using them as an anchor tenant in a new market.

In the case that you inadvertently create an unprofitable pricing model, there are some ways to mitigate the risks associated with them, including:

- Limit the use of the service through [usage limits](#).
- Require the use of capacity reservations.
- Request the tenant move to a higher feature or service tier.

Risky pricing models

When implementing a pricing model for a solution, you will usually have to make assumptions about how it will be used. If these assumptions turn out to be incorrect or the usage patterns change over time, then your pricing model may become unprofitable. Pricing models that are at risk of becoming unprofitable are known as *risky pricing* models. For example, if you adopt a pricing model that expects users to self-limit the amount that they use your solution, then you may have implemented a risky pricing model.

Most SaaS solutions will add new features regularly. This increases the ROV to users, which may also lead to increases in the amount the solution is used. This could result in a solution that becomes unprofitable, if the use of new features drives usage, but the pricing model doesn't factor this in.

For example, if you introduce a new video upload feature to your solution, which uses a consumption-based

resource, and user uptake of the feature is higher than expected, then consider a combination of [usage limits](#) and [feature and service level pricing](#).

Usage limits

Usage limits enable you to restrict the usage of your service in order to prevent your pricing models from becoming unprofitable, or to prevent a single tenant from consuming a disproportionate amount of the capacity of your service. This can be especially important in multitenant services, where a single tenant can impact the experience of other tenants by over-consuming resources.

NOTE

It's important to make your customers aware that you apply usage limits. If you implement usage limits without making your customers aware of the limit, then it will result in customer dissatisfaction. This means that it's important to identify and plan usage limits ahead of time. The goal should be to plan for the limit, and to then communicate the limits to customers before they become necessary.

Usage limits are often used in combination with [feature and service-level pricing](#), to provide a higher amount of usage at higher tiers. Limits are also commonly used to protect core components that will cause system bottlenecks or performance issues, if they are over-consumed.

Rate limits

A common way to apply a usage limit is to add rate limits to APIs or to specific application functions. This is also referred to as [throttling](#). Rate limits prevent continuous overuse. They are often used to limit the number of calls to an API, over a defined time period. For example, an API may only be called 20 times per minute, and it will return an HTTP 429 error, if it is called more frequently than this.

Some situations, where rate limiting is often used, include the following:

- REST APIs.
- Asynchronous tasks.
- Tasks that are not time sensitive.
- Actions that incur a high cost to execute.
- Report generation.

Implementing rate limiting can increase the solution complexity, but services like Azure API Management can make this simpler by applying [rate limit policies](#).

Pricing model lifecycle

Like any other part of your solution, pricing models have a lifecycle. As your application evolves over time, you might need to change your pricing models. This may be driven by changing customer needs, commercial requirements, or changes to functionality within your application. Some common pricing lifecycle changes include the following:

- Adding a completely new pricing model. For example, adding a [consumption pricing model](#) to a solution that currently offers a [flat rate model](#).
- Retiring an existing pricing model.
- Adding a tier to an existing pricing model.
- Retiring a tier in an existing pricing model.
- Changing usage limits on a feature in a pricing model.
- Adding or removing features from a [feature and service-level pricing model](#).
- Changing from a business-to-consumer (B2C) commercial model, to a business-to-business (B2B)

commercial model. This change then necessitates new pricing structures for your business customers.

It is usually complex to implement and manage many different pricing models at once. It's also confusing to your customers. So, it's better to implement only one or two models, with a small number of tiers. This makes your solution more accessible and easier to manage.

NOTE

Pricing models and billing functions should be tested, ideally using automated testing, just like any other part of your system. The more complex the pricing models, the more testing is required, and so the cost of development and new features will increase.

When changing pricing models, you will need to consider the following factors:

- Will tenants want to migrate to the new model?
- Is it easy for tenants to migrate to the new model?
- Will new pricing models expose your service to risk? For example, are you removing rate limits that are currently protecting critical resources from overuse?
- Do tenants have a clear path for migrating to the new pricing model?
- How will you prevent tenants from using older pricing models, so that you can retire them?
- Are tenants likely to receive *bill shock* (a negative reaction to an unexpected bill) upon changing pricing models?
- Are you monitoring the performance and utilization of your services, for new or changed pricing models, so that you can ensure continued profitability?
- Are you able to clearly communicate the ROV for new pricing models, to your existing tenants?

Next steps

Consider how you'll [measure consumption](#) by tenants in your solution.

Measure the consumption of each tenant

3/10/2022 • 6 minutes to read • [Edit Online](#)

As a solution provider, it's important to measure the consumption of each tenant in your multitenant solution. By measuring the consumption of each tenant, you can ensure that the cost of goods sold (COGS), for delivering the service to each tenant, is profitable. On this page, we provide guidance for technical decision-makers about the importance of measuring consumption, and the approaches you can consider to measure a tenant's consumption, as well as the tradeoffs involved.

There are two primary concerns driving the need for measuring each tenant's consumption:

- You need to measure the actual cost to serve each tenant. This is important to monitor the profitability of the solution for each tenant.
- You need to determine the amount to charge the tenant, when you're using [consumption-based pricing](#).

However, it can be difficult to measure the actual resources used by a tenant in a multitenant solution. Most services that can be used as part of a multitenant solution don't automatically differentiate or break down usage, based on whatever you define a tenant to be. For example, consider a service that stores data for all of your tenants in a single relational database. It's difficult to determine exactly how much space each tenant uses of that relational database, either in terms of storage or of the compute capacity that's required to service any queries and requests.

By contrast, for a single-tenant solution, you can use Azure Cost Management within the Azure portal, to get a complete cost breakdown for all the Azure resources that are consumed by that tenant.

Therefore, when facing these challenges, it is important to consider how to measure consumption.

NOTE

In some cases, it's commercially acceptable to take a loss on delivering service to a tenant, for example, when you enter a new market or region. This is a commercial choice. Even in these situations, it's still a good idea to measure the consumption of each tenant, so that you can plan for the future.

Indicative consumption metrics

Modern applications (built for the cloud) are usually made up of many different services, each with different measures of consumption. For example, a storage account measures consumption based on the amount of data stored, the data transmitted, and the numbers of transactions. However, Azure App Service consumption is measured by the amount of compute resources allocated over time. If you have a solution that includes a storage account and App Service resources, then combining all these measurements together to calculate the actual COGS (cost of goods sold) can be a very difficult task. Often, it is easier to use a single indicative measurement to represent consumption in the solution.

For example, in the case of a multitenant solution that shares a single relational database, you might determine that the data stored is a good indicative consumption metric.

NOTE

Even if you use the volume of data stored by a tenant as an indicative consumption measure, it might not be a true representation of consumption for every tenant. For example, if a particular tenant does a lot more reads or runs more reporting from the solution, but it doesn't write a lot of data, then it could use a lot more compute than the storage requirements would indicate.

It is important to occasionally measure and review the actual consumption across your tenants, to determine whether the assumptions you're making about your indicative metrics are correct.

Transaction metrics

An alternative way of measuring consumption is to identify a key transaction that is representative of the COGS for the solution. For example, in a document management solution, it could be the number of documents created. This can be useful, if there is a core function or feature within a system that is transactional, and if it can be easily measured.

This method is usually easy and cost effective to implement, as there is usually only a single point in your application that needs to record the number of transactions that occur.

Per-request metrics

In solutions that are primarily API-based, it might make sense to use a consumption metric that is based around the number of API requests being made to the solution. This can often be quite simple to implement, but it does require you to use APIs as the primary interface to the system. There will be an increased operational cost of implementing a per-request metric, especially for high volume services, because of the need to record the request utilization (for audit and billing purposes).

NOTE

User-facing solutions that consist of a single-page application (SPA), or a mobile application that uses the APIs, may not be a good fit for the per-request metric. This is because there is little understanding by the end user of the relationship between the use of the application and the consumption of APIs. Your application might be chatty (it makes many API requests) or chunky (it makes too few API requests), and the user wouldn't notice a difference.

WARNING

Make sure you store request metrics in a reliable data store that's designed for this purpose. For example, although Azure Application Insights can track requests and can even track tenant IDs (by using [properties](#)), Application Insights is not designed to store every piece of telemetry. It removes data, as part of its [sampling behavior](#). For billing and metering purposes, choose a data store that will give you a high level of accuracy.

Estimate consumption

When measuring the consumption for a tenant, it may be easier to provide an estimate of the consumption for the tenant, rather than trying to calculate the exact amount of consumption. For example, for a multitenant solution that serves many thousands of tenants in a single deployment, it is reasonable to approximate that the cost of serving the tenant is just a percentage of the cost of the Azure resources.

You might consider estimating the COGS for a tenant, in the following cases:

- You aren't using [consumption-based pricing](#).
- The usage patterns and cost for every tenant is similar, regardless of size.

- Each tenant consumes a low percentage (say, <2%), of the overall resources in the deployment.
- The per-tenant cost is low.

You might also choose to estimate consumption in combination with [indicative consumption metrics](#), [transaction metrics](#), or [per-request metrics](#). For example, for an application that primarily manages documents, the percentage of overall storage used by a tenant, to store its documents, gives a close enough representation of the actual COGS. This can be a useful approach, when measuring the COGS is difficult or when it would add too much complexity to the application.

On-charging your costs

In some solutions, you can simply charge your customers the COGS for their tenant's resources. For example, you might use [Azure resource tags](#) to allocate billable Azure resources to tenants. You can then determine the cost to each tenant for the set of resources that's dedicated to them, plus a margin for profit and operation.

NOTE

Some [Azure services don't support tags](#). For these services, you will need to attribute the costs to a tenant, based on the resource name, resource group, or subscription.

[Azure Cost Analysis](#) can be used to analyze Azure resource costs for single tenant solutions that use tags, resource groups, or subscriptions to attribute costs.

However, this becomes prohibitively complex in most modern multitenant solutions, because of the challenge of accurately determining the exact COGS to serve a single tenant. This method should only be considered for very simple solutions, solutions that have single-tenant resource deployments, or custom tenant-specific add-on features within a larger solution.

Some Azure services provide features that allow other methods of attribution of costs in a multitenant environment. For example, Azure Kubernetes Service supports [multiple node pools](#), where each tenant is allocated a node pool with [node pool tags](#), which are used to attribute costs.

Next steps

Consider the [update deployment model you will use](#).

Considerations for updating a multitenant solution

3/10/2022 • 7 minutes to read • [Edit Online](#)

One of the benefits of cloud technology is continuous improvement and evolution. As a service provider, you need to apply updates to your solution: you might need to make changes to your Azure infrastructure, your code/applications, your database schemas, or any other component. It's important to plan how you update your environments. In a multitenant solution, it's particularly important to be clear about your update policy, since some of your tenants may be reluctant to allow changes to their environments, or they might have requirements that limit the times when you can update their service. You need to identify your tenants' requirements, clarify your own requirements to operate your service, find a balance that works for everyone, and then communicate this clearly. On this page, we provide guidance for technical decision-makers about the approaches you can consider to update your tenants' software, and the tradeoffs involved.

Your customers' requirements

Consider the following questions:

- Do your customers have expectations or requirements about when they can be updated? These might be formally communicated to you in contracts or service-level agreements, or they might be informal.
- Do your customers expect service-defined or even self-defined maintenance windows? They might need to communicate to their own customers about any potential outages.
- Do your customers have any regulatory concerns that require additional approval before updates can be applied? For example, if you provide a health solution that includes IoT components, you might need to get approval from the United States Food and Drug Administration (FDA) before applying an update.
- Are any of your customers particularly sensitive or resistant to having updates applied? Try to understand why. For example, if they run a physical store or a retail website, they may want to avoid updates around Black Friday, as the risks are higher than potential benefits.
- What's your track record of successfully completing updates without any impact to your customers? You should follow good DevOps, testing, deployment, and monitoring practices to reduce the likelihood of outages, and to ensure that you quickly identify any issues that updates introduce. If your customers know that you're able to update their environments smoothly, they're less likely to object.
- Will customers want to roll back updates if there's a breaking change?

Your requirements

You also need to consider the following questions from your own perspective:

- Is it reasonable for your customers to have control over when updates are applied? If you're building a solution used by large enterprise customers, the answer may be yes. However, if you're building a consumer-focused solution, it's unlikely you'll give any control over how you upgrade or operate your solution.
- How many versions of your solution can you reasonably maintain at one time? Remember that if you find a bug and need to hotfix it, you might need to apply the hotfix to all of the versions in use.
- What are the consequences of letting customers fall too far behind the current version? If you release new features on a regular basis, will old versions become obsolete quickly? Also, depending on your upgrade strategy and the types of changes, you might need to maintain separate infrastructures for each version of your solution. So, there might be both operational and financial costs, as you maintain support for older versions.
- Can your deployment strategy support rollbacks to previous versions? Is this something you want to enable?

NOTE

Consider whether you need to take your solution offline for updates or maintenance. Generally, outage windows are seen as an outdated practice, and modern DevOps practices and cloud technologies enable you to avoid downtime during updates and maintenance. You need to design for this, so it's important to consider your update process when you're designing your solution architecture. Note that even if you don't plan for outages, you might still consider defining a regular maintenance window, so that your customers understand that changes happen during specific times. For more information on achieving zero-downtime deployments, see [Achieving no downtime through versioned service updates](#).

Find a balance

If you leave cadence of your service updates entirely to your tenants' discretion, they may choose to never update. It's important to allow yourself to update your solution, while factoring in any reasonable concerns or constraints that your customers might have. For example, if a customer is particularly sensitive to updates on a Friday (since that's their busiest day of the week), can you just as easily defer updates to Mondays, without impacting your solution?

One approach that can work well is to roll out updates on a tenant-by-tenant basis, using [one of the approaches described below](#). Give your customer notice of planned updates. Allow customers to temporarily opt out, but not forever; put a reasonable limit on when you will require the update to be applied.

Also, consider allowing yourself the ability to deploy security patches, or other critical hotfixes, with minimal or no advance notice.

Another approach can be to allow tenants to initiate their own updates, at a time of their choosing. Again, you should provide a deadline, at which point you apply the update on their behalf.

WARNING

Be careful about enabling tenants to initiate their own updates. This is complex to implement, and it will require significant development and testing effort to deliver and maintain.

Whatever you do, ensure you have a process to monitor the health of your tenants, especially before and after updates are applied. Often, critical production incidents (also called *live-site incidents*) happen after updates to code or configuration. Therefore, it's important you proactively monitor for and respond to any issues to retain customer confidence. For more information about monitoring, see [Monitoring for DevOps](#).

Communicate with your customers

Clear communication is key to building your customers' confidence. It's important to explain the benefits of regular updates, including new features, bug fixes, resolving security vulnerabilities, and performance improvements. One of the benefits of a modern cloud-hosted solution is the ongoing delivery of features and updates.

Consider the following questions:

- Will you notify customers of upcoming updates?
- If you do, will you implicitly request permission by providing an opt-out process?
- Do you have a scheduled maintenance window that you use when you apply updates?
- What if you have an emergency update, like a critical security patch? Can you force updates in those situations?
- If you can't proactively notify customer of upcoming updates, can you provide retrospective notifications? For example, can you update a page on your website with the list of updates that you've applied?

Communicate with your customer support team

It's important that your own support team has full visibility into updates that have been applied to each tenant. Customer support representatives should be able to easily answer the following questions:

- Have updates recently been applied to a tenant's infrastructure?
- What was the nature of those updates?
- What was the previous version?
- How frequently are updates applied to this tenant?

If one of your customers has a problem because of an update, you need to ensure your customer support team has the information necessary to understand what's changed.

Deployment strategies to support updates

Consider how you will deploy updates to your infrastructure. This is heavily influenced by the [tenancy model](#) that you use. Three common approaches for deploying updates are deployment stamps, feature flags, and deployment rings.

In all cases, ensure that you have sufficient reporting/visibility, so that you know what version of infrastructure, software, or feature each tenant is on, what they are eligible to migrate to, and any time-related data associated with those states.

Deployment Stamps pattern

Some multitenant applications are a good fit for the [Deployment Stamps pattern](#), in which you deploy multiple copies of your application and other components. Depending on your isolation requirements, you might deploy a stamp for each tenant, or shared stamps that run multiple tenants' workloads.

Stamps are a great way to provide isolation between tenants. They also provide you with flexibility for your update process, since you can roll out updates progressively across stamps, without affecting others.

Feature flags

[Feature flags](#) enable you to add functionality to your solution, while only exposing it to a subset of your customers or tenants. You might use feature flags, if you deploy updates regularly but want to avoid showing new functionality, or if you want to avoid applying changes in behavior until a customer opts in.

You can embed feature flag support into your application by writing code yourself, or by using a service like [Azure App Configuration](#).

Deployment rings

[Deployment rings](#) enable you to progressively roll out updates across a set of tenants or deployment stamps. You can assign a subset of tenants to each ring. A *canary* ring includes your own test tenants and customers who want to have updates as soon as they are available, with the understanding that they may receive more frequent updates, and that updates might not have been through a comprehensive validation process as in the other things. An *early adopter* ring contains tenants who are slightly more risk-averse, but who are still prepared to receive regular updates. Most of your tenants will belong to the *users* ring, which receives less frequent and more highly tested updates.

API versions

If your service exposes an external API, consider that any updates you apply might affect the way that customers or partners integrate with your platform. In particular, you need to be conscious of breaking changes to your APIs. Consider using [an API versioning strategy](#) to mitigate the risk of updates to your API.

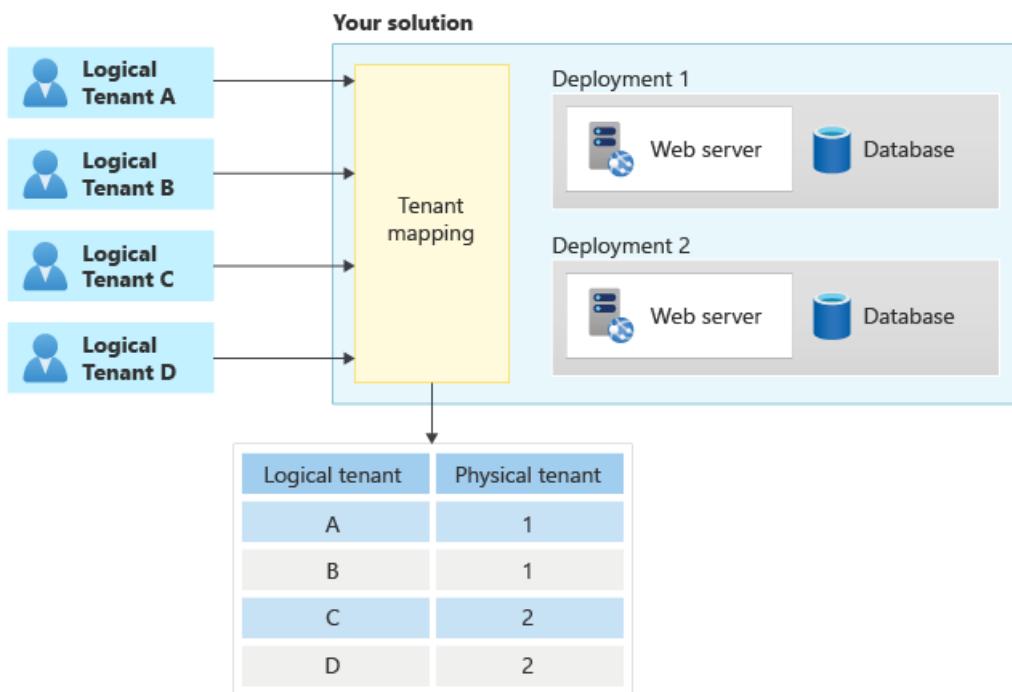
Next steps

- Consider when you would [map requests to tenants, in a multitenant solution](#).
- Review the [DevOps checklist](#) in Azure Well-Architected Framework.

Map requests to tenants in a multitenant solution

3/10/2022 • 9 minutes to read • [Edit Online](#)

Whenever a request arrives into your application, you need to determine the tenant that the request is intended for. When you have tenant-specific infrastructure that may even be hosted in different geographic regions, you need to match the incoming request to a tenant. Then, you must forward the request to the physical infrastructure that hosts that tenant's resources, as illustrated below:



On this page, we provide guidance for technical decision-makers about the approaches you can consider to map requests to the appropriate tenant, and the tradeoffs involved in the approaches.

NOTE

This page mostly discusses HTTP-based applications, like websites and APIs. However, many of same underlying principles apply to multitenant applications that use other communication protocols.

Approaches to identify tenants

There are multiple ways you can identify the tenant for an incoming request.

Domain names

If you use [tenant-specific domain or subdomain names](#), it's likely that requests can be easily mapped to tenants by using the `Host` header, or another HTTP header that includes the original hostname for each request.

However, consider the following questions:

- How will users know which domain name to use to access the service?
- Do you have a central entry point, like a landing page or login page, that all the tenants use? If you do, how will users identify the tenant that they need to access?
- What other information are you using to verify access to the tenant, such as authorization tokens? Do the authorization tokens include the tenant-specific domain names?

HTTP request properties

If you don't use tenant-specific domain names, you might still be able to use aspects of the HTTP request to identify the tenant that a particular request is for. For example, consider an HTTP request that identifies the tenant name as `tailwindtraders`. This might be communicated using the following:

- **The URL path structure**, such as `https://app.contoso.com/tailwindtraders/`.
- **A query string** in the URL, such as `https://contoso.com/app?tenant=tailwindtraders`.
- **A custom HTTP request header**, such as `X-Tenant-Id: tailwindtraders`.

IMPORTANT

Custom HTTP request headers aren't useful where HTTP GET requests are issued from a web browser, or where the requests are handled by some types of web proxy. You should only use custom HTTP headers for GET operations when you're building an API, or if you control the client that issues the request and there's no web proxy included in the request processing chain.

When using this approach, you should consider the following questions:

- Will users know how to access the service? For example, if you use a query string to identify tenants, will a central landing page need to direct users to the correct tenant, by adding the query string?
- Do you have a central entry point, like a landing page or login page, that all tenants use? If you do, how will users identify the tenant that they need to access?
- Does your application provide APIs? For example, is your web application a single-page application (SPA) or a mobile application with an API backend? If it is, you might be able to use an [API gateway](#) or [reverse proxy](#) to perform tenant mapping.

Token claims

Many applications use claims-based authentication and authorization protocols, such as OAuth 2.0 or SAML. These protocols provide authorization tokens to the client. A token contains a set of *claims*, which are pieces of information about the client application or user. Claims can be used to communicate information like a user's email address. Your system can then include the user's email address, look up the user-to-tenant mapping, and then forward the request to the appropriate physical tenant infrastructure. Or, you might even include the tenant mapping in your identity system, and add a tenant ID claim to the token.

If you are using claims to map requests to tenants, you should consider the following questions:

- Will you use a claim to identify a tenant? Which claim will you use?
- Can a user be a member of multiple tenants? If this is possible, then how will users select the tenants they'd like to work with?
- Is there a central authentication and authorization system for all tenants? If not, how will you ensure that all token authorities issue consistent tokens and claims?

API keys

Many applications expose APIs. These might be for internal use within your organization, or for external use by partners or customers. A common method of authentication for APIs is to use an *API key*. API keys are provided with each request, and they can be used to look up the tenant.

API keys can be generated in several ways. A common approach is to generate a cryptographically random value and store it in a lookup table, alongside the tenant ID. When a request is received, your system finds the API key in the lookup table, and it then matches it to a tenant ID. Another approach is to create a meaningful string with a tenant ID included inside it, and then you would digitally sign the key, by using an approach like [HMAC](#). When you process each request, you verify the signature and then extract the tenant ID.

NOTE

API keys don't provide a high level of security because they need to be manually created and managed, and because they don't contain claims. A more modern and secure approach is to use a claims-based authorization mechanism with short-lived tokens, such as OAuth 2.0 or OpenID Connect.

Consider the following questions:

- How will you generate and issue API keys?
- How will your API clients securely receive and store the API key?
- Do you need your API keys to expire after a period of time? How will you rotate your clients' API keys, without causing downtime?
- Does just relying on customer-rolled API keys provide an adequate level of security for your APIs?

NOTE

API keys are not the same as passwords. API keys must be generated by the system, and they must be unique across all the tenants, so that each API key can be uniquely mapped to a single tenant. API gateways, such as [Azure API Management](#), can generate and manage API keys, validate keys on incoming requests, and map keys to individual API subscribers.

Client certificates

Client certificate authentication, sometimes called mutual TLS (mTLS), is commonly used for service-to-service communication. Client certificates provide a secure way to authenticate clients. Similarly to tokens and claims, client certificates provide *attributes* that can be used to determine the tenant. For example, the *subject* of the certificate may contain the email address of the user, which can be used to look up the tenant.

When planning to use client certificates for tenant mapping consider the following:

- How will you safely issue and renew the client certificates that are trusted by your service? Client certificates can be complex to work with, since they require special infrastructure to manage and issue certificates.
- Will client certificates be used only for initial login requests, or attached to all requests to your service?
- Will the process of issuing and managing certificates become unmanageable when you have a large number of clients?
- How will you implement the mapping between the client certificate and the tenant?

Reverse proxies

A reverse proxy, also referred to as an application proxy, can be used to route HTTP requests. A reverse proxy accepts a request from an ingress endpoint, and it can forward the request to one of many backend endpoints. Reverse proxies are useful for multitenant applications since they can perform the mapping between some piece of request information, offloading the task from your application infrastructure.

Many reverse proxies can use the properties of a request to make a decision about tenant routing. They can inspect the destination domain name, URL path, query string, HTTP headers, and even claims within tokens.

The following common reverse proxies are used in Azure:

- [Azure Front Door](#) is a global load balancer and web application firewall. It uses the Microsoft global edge network to create fast, secure, and highly scalable web applications.
- [Azure Application Gateway](#) is a managed web traffic load balancer that you deploy into the same physical region as your backend service.
- [Azure API Management](#) is optimized for API traffic.

- Commercial and open-source technologies (that you host yourself) include nginx, Traefik, and HAProxy.

Request validation

It is important that your application validates that any requests that it receives are authorized for the tenant. For example, if your application uses a custom domain name to map requests to the tenant, then your application must still check that each request received by the application is authorized for that tenant. Even though the request includes a domain name or other tenant identifier, it doesn't mean you should automatically grant access. When you use OAuth 2.0, you perform the validation by inspecting the *audience* and *scope* claims.

NOTE

This is part of the *assume zero trust* security design principle in the [Microsoft Azure Well-Architected Framework](#). When implementing request validation, you should consider the following:

- How will you authorize all the requests to your application? You need to authorize requests, regardless of the approach you use to map them to physical infrastructure.
- Use trusted and widely used authentication and authorization frameworks and middleware, instead of implementing all of the validation logic yourself. For example, don't build token signature validation logic or client certificate cryptography libraries. Instead, use features of your application platform (or known trusted packages) that have been validated and tested.

Performance

Tenant mapping logic likely runs on every request to your application. Consider how well the tenant mapping process will scale, as your solution grows. For example, if you query a database table as part of your tenant mapping, will the database support a large amount of load? If your tenant mapping requires decrypting a token, will the computation requirements become too high over time? If your traffic is fairly modest, then this isn't likely to affect your overall performance. When you have a high-scale application, though, the overhead involved in this mapping can become significant.

Session affinity

One approach to reducing the performance overhead of tenant mapping logic is to use *session affinity*. Rather than perform the mapping on every request, consider computing the information only on the first request for each session. Your application then provides a *session cookie* to the client that can then be passed back to your service, with all subsequent client requests within that session.

NOTE

Many networking and application services in Azure can issue session cookies and natively route requests by using session affinity.

Consider the following questions:

- Can you use session affinity to reduce the overhead of mapping requests to tenants?
- What services do you use to route requests to the physical deployments for each tenant? Do these services support cookie-based session affinity?

Tenant migration

Tenants often need to be moved to new infrastructure as part of the [tenant lifecycle](#). When a tenant is moved to a new deployment, the HTTP endpoints they access might change. When this happens, consider that your tenant-mapping process needs to be updated. You may need to consider the following:

- If your application uses domain names for mapping requests, then it might also require a DNS change at the time of the migration. The DNS change might take time to propagate to clients, depending on the time-to-live of the DNS entries in your DNS service.
- If your migration changes the addresses of any endpoints during the migration process, then consider temporarily redirecting requests for the tenant to a maintenance page that automatically refreshes.

Next steps

Learn about [considerations when you work with domain names in a multitenant application](#).

Considerations when using domain names in a multitenant solution

3/10/2022 • 9 minutes to read • [Edit Online](#)

In many multitenant web applications, a domain name can be used as a way to identify a tenant, to help with routing requests, and to provide a branded experience to your customers. Two common approaches are to use subdomains and custom domain names. On this page, we provide guidance for technical decision-makers about the approaches you can consider and their tradeoffs.

Subdomains

Each tenant might get a unique subdomain under a common shared domain name. Let's consider an example multitenant solution built by Contoso. Customers purchase Contoso's product to help manage their invoice generation. All of Contoso's tenants might be assigned their own subdomain, under the `contoso.com` domain name. Or, if you use regional deployments, you might assign subdomains under the `us.contoso.com` and `eu.contoso.com` domains. In this article, we refer to these as *stem domains*. Each customer gets their own subdomain under your stem domain. For example, Tailwind Toys might be assigned `tailwind.contoso.com`, and Adventure Works might be assigned `adventureworks.contoso.com`.

NOTE

Many Azure services use this approach. For example, when you create an Azure storage account, it is assigned a set of subdomains for you to use, such as `<your account name>.blob.core.windows.net`.

Manage your domain namespace

When you create subdomains under your own domain name, you need to be mindful that you could have multiple customers with similar names. Since they share a single stem domain, the first customer to get a particular domain will get their preferred name, and subsequent customers will have to use alternate subdomain names, since full domain names must be globally unique.

Wildcard DNS

Consider using wildcard DNS entries to simplify the management of subdomains. Instead of creating DNS entries for `tailwind.contoso.com`, `adventureworks.contoso.com`, and so forth, you could instead create a wildcard entry for `*.contoso.com` and direct all subdomains to single IP address (A record) or canonical name (CNAME record).

NOTE

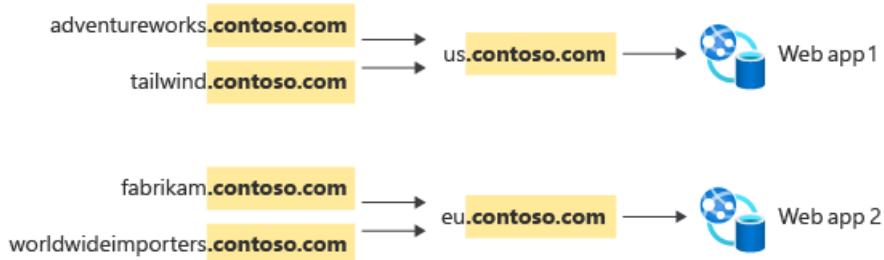
Make sure that your web-tier services support wildcard DNS, if you plan to rely on this feature. Many Azure services, including Azure Front Door and Azure App Service, support wildcard DNS entries.

Subdomains with multipart stem domains

Many multitenant solutions are spread across multiple physical deployments. This is common, when you need to comply with data residency requirements, or when you want to provide better performance by deploying resources geographically closer to the users. Even within a single region, you might also need to spread your tenants across independent deployments, as part of your scaling strategy. If you plan to use subdomains for each tenant, you might consider a multipart subdomain structure.

Here's an example: Contoso publishes a multitenant application for its four customers. Adventure Works and Tailwind Traders are in the United States, and their data is stored on a shared US instance of the Contoso platform. Fabrikam and Worldwide Importers are in Europe, and their data is stored on a European instance.

If Contoso chose to use a single stem domain, `contoso.com`, for all their customers, here's what this might look like:

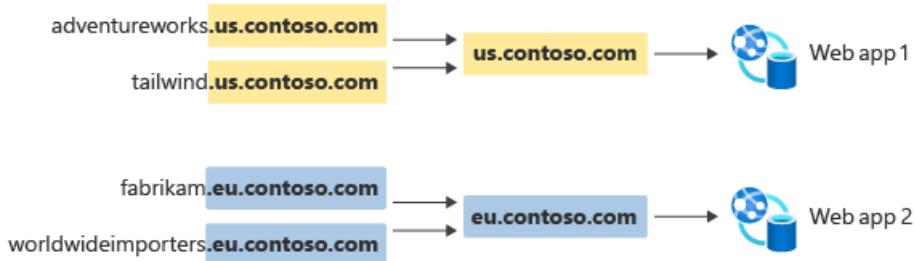


The DNS entries (that are required to support this configuration) might look like this:

SUBDOMAIN	CNAME TO
adventureworks.contoso.com	us.contoso.com
tailwind.contoso.com	us.contoso.com
fabrikam.contoso.com	eu.contoso.com
worldwideimporters.contoso.com	eu.contoso.com

Each new customer that is onboarded requires a new subdomain, and the number of subdomains grows with each customer.

Alternatively, Contoso could use deployment- or region-specific stem domains, like this:



The DNS entries for this deployment might look like this:

SUBDOMAIN	CNAME TO
*.us.contoso.com	us.contoso.com
*.eu.contoso.com	eu.contoso.com

Contoso doesn't need to create subdomain records for every customer. Instead, they have a single wildcard DNS record for each geography's deployment, and any new customers who are added underneath that stem will automatically inherit the CNAME record.

There are benefits and drawbacks to each approach. When using a single stem domain, each tenant you onboard requires a new DNS record to be created, which introduces more operational overhead. However, you have more flexibility, if you need to move tenants between deployments, since you can change the CNAME

record to direct their traffic to another deployment. This won't affect any other tenants. When using multiple stem domains, there's a lower management overhead, and you can reuse customer names across multiple regional stem domains, since each one effectively represents its own namespace.

Custom domain names

You might want to enable your customers to bring their own domain names. Some customers see this as an important aspect of their branding. It might also be required to meet customers' security requirements, especially if they need to supply their own TLS certificates. While it might seem trivial to enable customers to bring their own domain names, there are some hidden complexities to this approach, and it requires thoughtful consideration.

Name resolution

Ultimately, each domain name needs to be resolved to an IP address. As you've seen, the approach by which this happens can depend on whether you deploy a single instance or multiple instances of your solution.

Let's return to our example. One of Contoso's customers, Fabrikam, has asked to use `invoices.fabrikam.com`, as the custom domain name to access Contoso's service. Since Contoso has multiple deployments of their platform, they decide to use subdomains and CNAME records to achieve their routing requirements. Contoso and Fabrikam configure the following DNS records:

NAME	RECORD TYPE	VALUE	CONFIGURED BY
<code>invoices.fabrikam.com</code>	CNAME	<code>fabrikam.eu.contoso.com</code>	Fabrikam
<code>*.eu.contoso.com</code>	CNAME	<code>eu.contoso.com</code>	Contoso
<code>eu.contoso.com</code>	A	(Contoso's IP address)	Contoso

From a name resolution perspective, this chain of records accurately resolves requests for `invoices.fabrikam.com`, to the IP address of Contoso's European deployment.

Host header resolution

Name resolution is only half of the problem. All of the web components (within Contoso's European deployment) need to be aware of how to handle requests that arrive with Fabrikam's domain name in their `Host` request header. Depending on the specific web technologies that Contoso uses, this may require further configuration for each tenant's domain name, which adds extra operational overhead to the onboarding of tenants.

You can also consider rewriting host headers, so that regardless of the incoming request's `Host` header, your web server sees a consistent header value. For example, Azure Front Door enables you to rewrite `Host` headers, so that regardless of the request, your application server receives a single `Host` header. Azure Front Door propagates the original host header in the `X-Forwarded-Host` header, so that your application can inspect it, to resolve the tenant.

Domain validation

It's important to validate the ownership of custom domains before onboarding them. Otherwise, you risk a customer accidentally or maliciously *parking* a domain name.

Let's consider Contoso's onboarding process for Adventure Works, who have asked to use `invoices.adventureworks.com` as their custom domain name. Unfortunately, somebody made a typo when they tried to onboard the custom domain name, and they missed the *s*. So, they set it up as `invoices.adventurework.com`. Not only does the traffic not flow correctly, but when another company named *Adventure Work* tries to add their custom domain to Contoso's platform, they're told the domain name is

already in use.

When working with custom domains, especially within a self-service or automated process, it's common to require a domain verification step. This might require that the CNAME records be set up before the domain can be added. Alternatively, Contoso might generate a random string and ask Adventure Works to add a DNS TXT record with the string value. That would prevent the domain name from being added, until the verification is completed.

Dangling DNS and subdomain takeover attacks

When you work with custom domain names, you are potentially vulnerable to a class of attack called *dangling DNS or subdomain takeover*. This attack happens when customers disassociate their custom domain name from your service, but they don't delete the record from their DNS server. This DNS entry then points to a non-existent resource and is vulnerable to a takeover.

Let's consider how Fabrikam's relationship with Contoso might change:

1. Fabrikam has decided to no longer work with Contoso, and so they have terminated their business relationship.
2. Contoso has offboarded the Fabrikam tenant, and they requested for `fabrikam.contoso.com` to no longer work. However, Fabrikam forgot to delete the CNAME record for `invoices.fabrikam.com`.
3. A malicious actor creates a new Contoso account and gives it the name `fabrikam`.
4. The attacker onboards the custom domain name `invoices.fabrikam.com` to their new tenant. Since Contoso performs CNAME-based domain validation, they check Fabrikam's DNS server. They see that the DNS server returns a CNAME record for `invoices.fabrikam.com`, which points to `fabrikam.contoso.com`. Contoso considers the custom domain validation to be successful.
5. If any Fabrikam employees tried to access the site, requests would appear to work. If the attacker sets up their Contoso tenant with Fabrikam's branding, employees might be fooled into accessing the site and providing sensitive data, which the attacker can then access.

A common strategy to protect against dangling DNS attacks is to require that the CNAME record is deleted, before the domain name can be removed from the tenant's account. You could also consider prohibiting the reuse of tenant identifiers, and you can then strengthen your custom domain onboarding process by using a randomly generated TXT record (that is different for each onboarding attempt).

TLS/SSL certificates

Transport Layer Security (TLS) is an essential component, when working with modern applications. It provides trust and security to your web applications. The ownership and management of TLS certificates is something that needs careful consideration, for multitenant applications.

Typically, the owner of a domain name will be responsible for issuing and renewing its certificates. For example, Contoso is responsible for issuing and renewing TLS certificates for `us.contoso.com`, as well as a wildcard certificate for `*.contoso.com`. Similarly, Fabrikam would generally be responsible for managing any records for the `fabrikam.com` domain, including `invoices.fabrikam.com`. The CAA (Certificate Authority Authorization) DNS record type can be used by a domain owner, to ensure that only specific authorities can create certificates for their domain.

If you plan to allow customers to bring their own domains, consider whether you plan to issue the certificate on the customer's behalf, or whether the customers must bring their own certificates. Each option has benefits and drawbacks. If you issue a certificate for a customer, you can handle the renewal of the certificate, so the customer doesn't have to remember to keep it updated. However, if the customers have CAA records on their domain names, they might need to authorize you to issue certificates on their behalf. If you expect customers should issue and provide you with their own certificates, you are responsible for receiving and managing the private keys in a secure manner, and you might have to remind your customers to renew the certificate before it

expires, to avoid an interruption in their service.

Several Azure services support automatic management of certificates for custom domains. For example, Azure Front Door and App Service provide certificates for custom domains, and they automatically handle the renewal process. This removes the burden of managing certificates, from your operations team. However, you still need to consider the question of ownership and authority, such as whether CAA records are in effect and configured correctly. Also, you need to ensure your customers' domains are configured to allow the certificates that are managed by the platform.

Next steps

Return to the [architectural considerations overview](#). Or, review the [Microsoft Azure Well-Architected Framework](#).

Architectural approaches for multitenancy

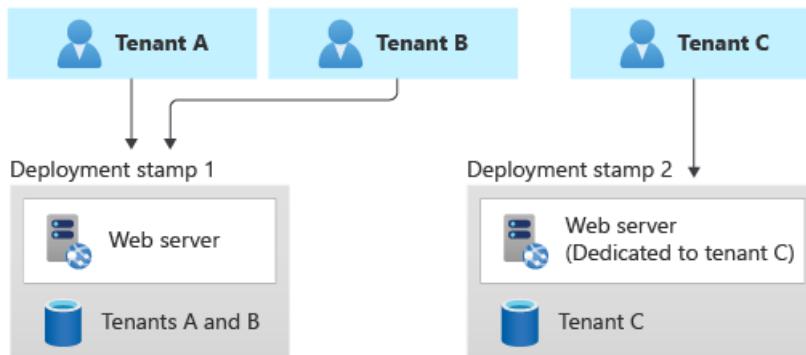
3/10/2022 • 3 minutes to read • [Edit Online](#)

There are many different ways that you can design and build multitenant solutions in Azure. At one extreme, you can share every resource in your solution between every tenant. At the other extreme, you can deploy isolated resources for every tenant. It might seem simple to deploy separate resources for every tenant, and it can work for a small numbers of tenants. However, it typically doesn't provide cost effectiveness, and it can become difficult to manage your resources. There are also various approaches that fit between these extremes, and they all have tradeoffs: scale, isolation, cost efficiency, performance, implementation complexity, and manageability.

Throughout this section, we discuss the main categories of Azure services that comprise a solution, including [compute](#), [storage and data](#), [networking](#), [deployment](#), [messaging](#), and [artificial intelligence and machine learning](#). Coming soon, we will also provide guidance for identity and IoT. For each category, we outline the key patterns and approaches you can consider when you're designing a multitenant solution, and some antipatterns to avoid.

Deployment Stamps pattern

The [Deployment Stamps pattern](#) is frequently used in multitenant solutions. It involves deploying dedicated infrastructure for a tenant or for a group of tenants. A single stamp might contain multiple tenants or might be dedicated to a single tenant.



When using single-tenant stamps, the Deployment Stamps pattern tends to be straightforward to implement, because each stamp is likely to be unaware of any other, so no multitenancy logic or capabilities need to be built into the application layer. When each tenant has their own dedicated stamp, this pattern provides the highest degree of isolation, and it mitigates the [Noisy Neighbor problem](#). It also provides the option for tenants to be configured or customized according to their own requirements, such as to be located in a specific geopolitical region or to have specific high availability requirements.

When using multitenant stamps, other patterns need to be considered to manage multitenancy within the stamp, and the Noisy Neighbor problem still might apply. However, by using the Deployment Stamps pattern, you can continue to scale as your solution grows.

The biggest problem with the Deployment Stamps pattern, when being used to serve a single tenant, tends to be the cost of the infrastructure. When using single-tenant stamps, each stamp needs to have its own separate set of infrastructure, which isn't shared with other tenants. You also need to ensure that the resources deployed for a stamp are sufficient to meet the peak load for that tenant's workload. Ensure that your [pricing model](#) offsets the cost of deployment for the tenant's infrastructure.

Single-tenant stamps often work well when you have a small number of tenants. As your number of tenants grows, it's possible but increasingly difficult to manage a fleet of stamps ([see this case study as an example](#)). You can also apply the Deployment Stamps pattern to create a fleet of multitenant stamps, which can provide

benefits for resource and cost sharing.

To implement the Deployment Stamps pattern, it's important to use automated deployment approaches. Depending on your deployment strategy, you might consider managing your stamps within your deployment pipelines, by using declarative infrastructure as code, such as Bicep, ARM templates, or Terraform templates. Alternatively, you might consider building custom code to deploy and manage each stamp, such as by using [Azure SDKs](#).

Intended audience

The pages in this section are intended to be useful for solution architects and lead developers of multitenant applications, including independent software vendors (ISVs) and startups who develop SaaS solutions. Much of the guidance in this section is generic and applies to multiple Azure services within a category.

Next steps

We recommend you review the [approaches for resource organization in a multitenant solution](#) before reviewing the guidance about specific categories of Azure services.

Azure resource organization in multitenant solutions

3/10/2022 • 13 minutes to read • [Edit Online](#)

Azure provides many options for organizing your resources. In a multitenant solution, there are specific tradeoffs to consider, when you plan your resource organization strategy. In this article, we review two core elements of organizing your Azure resources: tenant isolation and scale-out across multiple resources. We also describe how to work with Azure's resource limits and quotas, and how to scale your solution beyond these limits.

Key considerations and requirements

Tenant isolation requirements

When you deploy a multitenant solution in Azure, you need to decide whether you dedicate resources to each tenant or share resources between multiple tenants. Throughout the multitenancy approaches and [service-specific guidance](#) sections of this series, we describe the options and trade-offs for many categories of resources. In general, there are a range of options for *tenant isolation*. Review [Tenancy models to consider for a multitenant solution](#) for more guidance about how to decide on your isolation model.

Scale

Most Azure resources, as well as resource groups and subscriptions, impose limits that can affect your ability to scale. You might need to consider *scaling out* or *bin packing* to meet your planned number of tenants or your planned system load.

If you know with certainty that you won't grow to large numbers of tenants or to a high load, don't overengineer your scale-out plan. But if you plan for your solution to grow, carefully consider your scale-out plan. Ensure that you architect for scale, by following the guidance in this article.

If you have an automated deployment process and need to scale across resources, determine how you'll deploy and assign tenants across multiple resource instances. For example, how will you detect that you're approaching the number of tenants that can be assigned to a specific resource? Will you plan to deploy new resources *just in time* for when you need them? Or, will you deploy a pool of resources *ahead of time* so they're ready for you to use when you need them?

TIP

In the early stages of design and development, you might not choose to implement an automated scale-out process. You should still consider and clearly document the processes required to scale as you grow.

It's also important to avoid making assumptions in your code and configuration, which can limit your ability to scale. For example, you might need to scale out to multiple storage accounts. Ensure your application tier doesn't assume that it only connects to a single storage account for all tenants.

Approaches and patterns to consider

Tenant isolation

Azure resources are deployed and managed through a hierarchy. Most *resources* are deployed into [resource groups](#), which are contained in *subscriptions*. [Management groups](#) logically group subscriptions together. All of these hierarchical layers are associated with an [Azure Active Directory \(Azure AD\) tenant](#).

When you determine how to deploy resources for each tenant, you might isolate at different levels in the

hierarchy. Each option is valid for certain types of multitenant solutions, and comes with benefits and tradeoffs. It's also common to combine approaches, using different isolation models for different components of a solution.

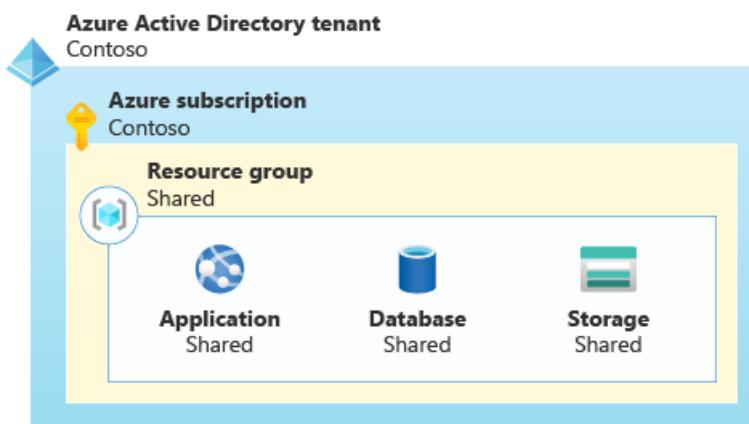
Isolation within a shared resource

You might choose to share an Azure resource among multiple tenants, and run all of their workloads on a single instance. Review the [service-specific guidance](#) for the Azure services you use to understand any specific considerations or options that might be important.

When you run single instances of a resource, you need to consider any service limits, subscription limits, or quotas that might be reached as you scale. For example, there's a maximum number of nodes that are supported by an Azure Kubernetes Service (AKS) cluster, and there's an upper limit on the number of transactions per second that are supported by a storage account. Consider how you'll [scale to multiple shared resources](#) as you approach these limits.

You also need to ensure your application code is fully aware of multitenancy, and that it restricts access to the data for a specific tenant.

As an illustration of the shared resource approach, suppose Contoso is building a multitenant SaaS application that includes a web application, a database, and a storage account. They might decide to deploy shared resources, and they'd use these resources to service all of their customers. In the following diagram, a single set of resources is shared by all the customers.



Separate resources in a resource group

You can also deploy dedicated resources for each tenant. You might deploy an entire copy of your solution for a single tenant. Or, you might share some components between tenants and deploy other components that are dedicated to a specific tenant.

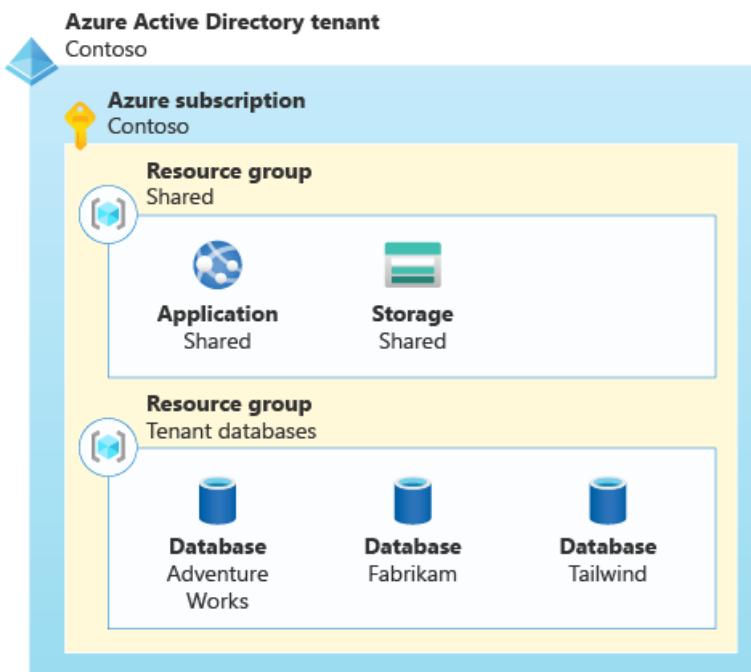
We recommend that you use resource groups to manage resources with the same lifecycle. In some multitenant systems, it makes sense to deploy resources for multiple tenants into a single resource group or a set of resource groups.

It's important that you consider how you deploy and manage these resources, including [whether the deployment of tenant-specific resources is initiated by your deployment pipeline or your application](#). You also need to determine how you'll [clearly identify that specific resources relate to specific tenants](#). Consider using a clear [naming convention strategy](#), [resource tags](#), or a tenant catalog database.

It's a good practice to use separate resource groups for the resources you share between multiple tenants and the resources that you deploy for individual tenants. However, for some resources, [Azure limits the number of resources of a single type that can be deployed into a resource group](#). This limit means you might need to [scale across multiple resource groups](#) as you grow.

Suppose Contoso has three customers: Adventure Works, Fabrikam, and Tailwind. They might choose to share the web application and storage account between the three customers, and then deploy individual databases for each tenant. In the following diagram, each customer is assigned a resource group that contains shared

resources and a resource group that contains a database.



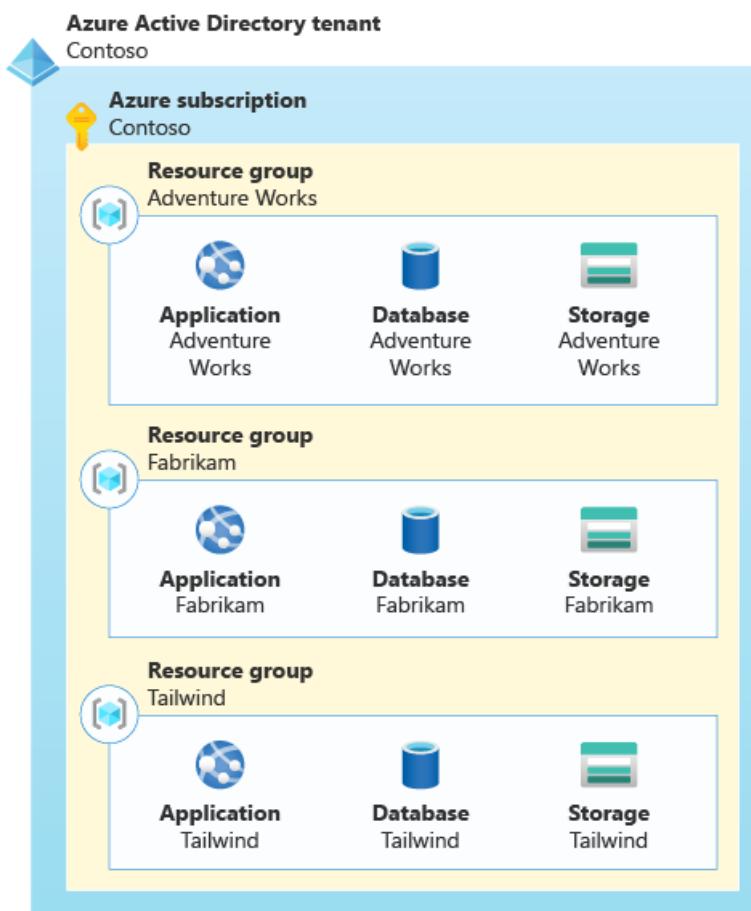
Separate resource groups in a subscription

When you deploy a set of resources for each tenant, consider using dedicated tenant-specific resource groups. For example, when you follow the [Deployment Stamps pattern](#), each stamp should be deployed into its own resource group. You can consider deploying multiple tenant-specific resource groups into a shared Azure subscription, which enables you to easily configure policies and access control rules.

You might choose to create a set of resource groups for each tenant, and also shared resource groups for any shared resources.

When you deploy tenant-specific resource groups into shared subscriptions, be aware of the maximum number of resource groups in each subscription, and other subscription-level limits that apply to the resources you deploy. As you approach these limits, you might need to [scale across multiple subscriptions](#).

In our example, Contoso might choose to deploy a stamp for each of their customers and place the stamps in dedicated resource groups within a single subscription. In the following diagram, a subscription, which contains three resource groups, is created for each customer.



Separate subscriptions

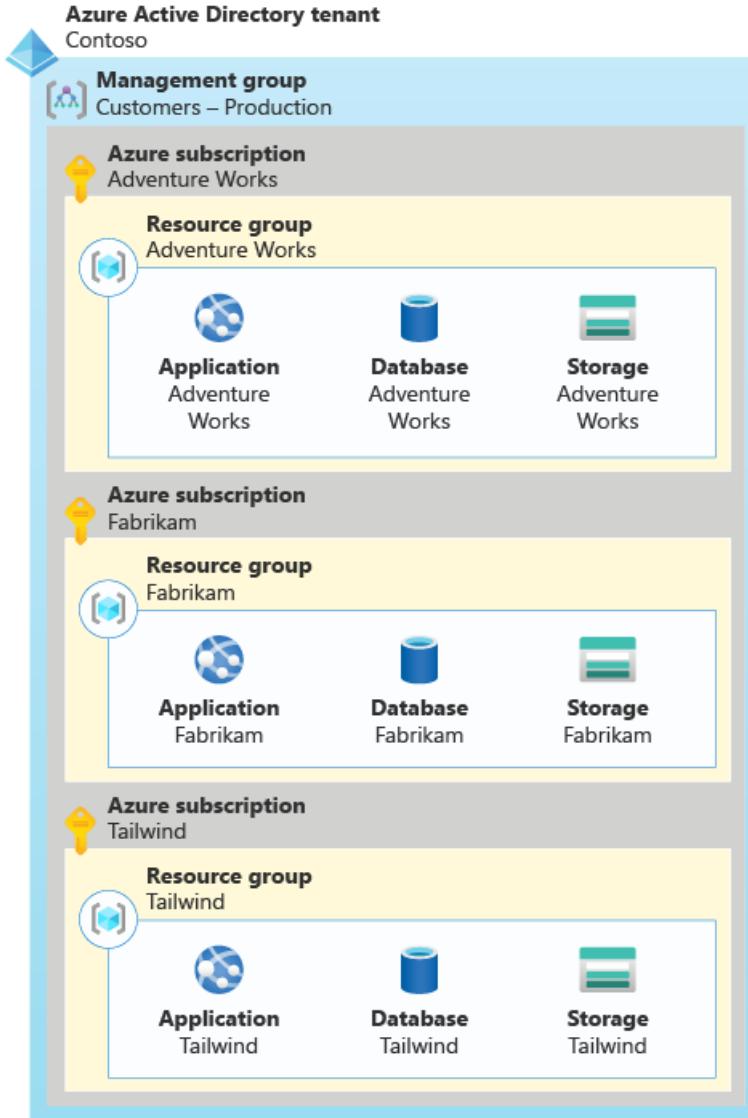
By deploying tenant-specific subscriptions, you can completely isolate tenant-specific resources. Additionally, because most quotas and limits apply within a subscription, using a separate subscription per tenant ensures that each tenant has full use of any applicable quotas. For some Azure billing account types, [you can programmatically create subscriptions](#). You can also use [Azure reservations](#) across subscriptions.

Make sure you are aware of the number of subscriptions that you can create. The maximum number of subscriptions might differ, depending on your commercial relationship with Microsoft or a Microsoft partner, such as if you have an [enterprise agreement](#).

However, it can be more difficult to request quota increases, when you work across a large number of subscriptions. The [Quota API](#) provides a programmatic interface for some resource types. However, for many resource types, quota increases must be requested by [initiating a support case](#). It can also be challenging to work with Azure support agreements and support cases, when you work with many subscriptions.

Consider grouping your tenant-specific subscriptions into a [management group](#) hierarchy, to enable easy management of access control rules and policies.

For example, suppose Contoso decided to create separate Azure subscriptions for each of their three customers, as shown in the following diagram. Each subscription contains a resource group, with the complete set of resources for that customer.



Each subscription contains a resource group, with the complete set of resources for that customer.

They use a management group to simplify the management of their subscriptions. By including *Production* in the management group's name, they can clearly distinguish any production tenants from non-production or test tenants. Non-production tenants would have different Azure access control rules and policies applied.

All of their subscriptions are associated with a single Azure Active Directory tenant. Using a single Azure AD tenant means that the Contoso team's identities, including users and service principals, can be used throughout their entire Azure estate.

Separate subscriptions in separate Azure AD tenants

It's also possible to manually create individual Azure Active Directory (Azure AD) tenants for each of your tenants, or to deploy your resources into subscriptions within your customers' Azure AD tenants. However, working with multiple Azure AD tenants makes it more difficult to authenticate, to manage role assignments, to apply global policies, and to perform many other management operations.

WARNING

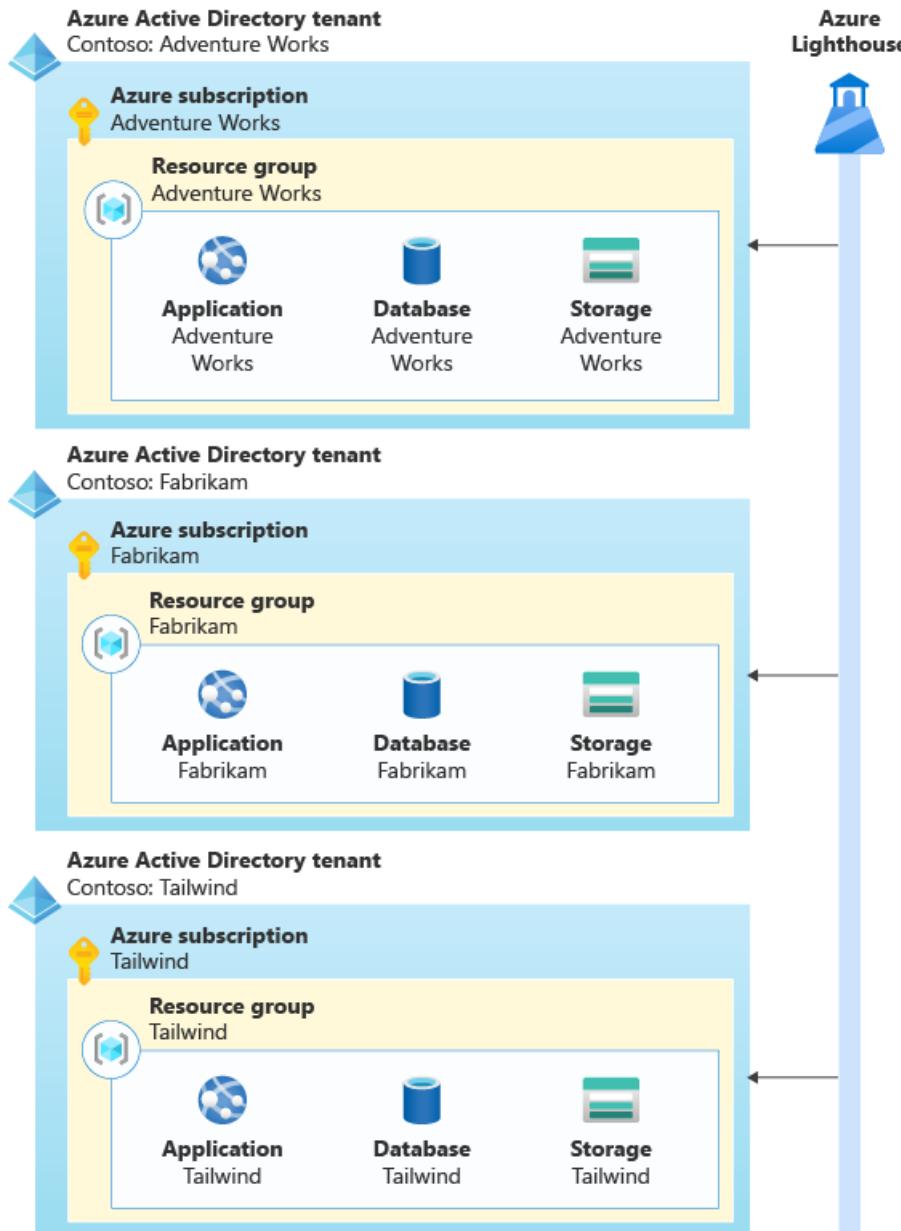
We advise against creating multiple Azure Active Directory tenants for most multitenant solutions.

Working across Azure AD tenants introduces extra complexity and reduces your ability to scale and manage your resources. Typically, this approach is only used by managed service providers (MSPs), who operate Azure environments on behalf of their customers.

A single Azure AD tenant can be used by multiple separate subscriptions and Azure resources. Before you make an effort to deploy multiple Azure AD tenants, consider whether there are other approaches that could achieve your purposes.

In situations where you need to manage Azure resources in subscriptions that are tied to multiple Azure AD tenants, consider using [Azure Lighthouse](#) to help manage your resources across your Azure AD tenants.

For example, Contoso could create separate Azure AD tenants and separate Azure subscriptions for each of their customers, as shown in the following diagram.



An Azure AD tenant is configured for each of Contoso's tenants, which contains a subscription and the resources required. Azure Lighthouse is connected to each Azure AD tenant.

Bin packing

Regardless of your resource isolation model, it's important to consider when and how your solution will scale out across multiple resources. You might need to scale your resources, as the load on your system increases, or as the number of tenants grows. Consider *bin packing* to deploy an optimal number of resources for your requirements.

TIP

In many solutions, it's easier to scale your entire set of resources together, instead of scaling resources individually. Consider following the [Deployment Stamps pattern](#).

Azure resources have [limits and quotas](#) that must be considered in your solution planning. For example, resources might support a maximum number of concurrent requests or tenant-specific configuration settings.

The way you configure and use each resource also affects the scalability of that resource. For example, given a certain amount of compute resources, your application can successfully respond to a defined number of transactions per second. Beyond this point, you might need to scale out. Performance testing helps you to identify the point at which your resources no longer meet your requirements.

NOTE

The principle of scaling to multiple resources applies even when you work with services that support multiple instances.

For example, Azure App Service supports scaling out the number of instances of your plan, but there are limits for how far you can scale a single plan. In a high-scale multitenant app, you might exceed these limits and need to deploy additional App Service resources to match your growth.

When you share some of your resources between tenants, you should first determine the number of tenants that the resource supports, when it's configured according to your requirements. Then, deploy as many resources as you need to serve your total number of tenants.

For example, suppose you deploy an Azure Application Gateway, as part of a multitenant SaaS solution. You review your application design, test the application gateway's performance under load, and review its configuration. Then, you determine that a single application gateway resource can be shared among 100 customers. According to your organization's growth plan, you expect to onboard 150 customers in your first year, so you need to plan to deploy multiple application gateways to service your expected load.

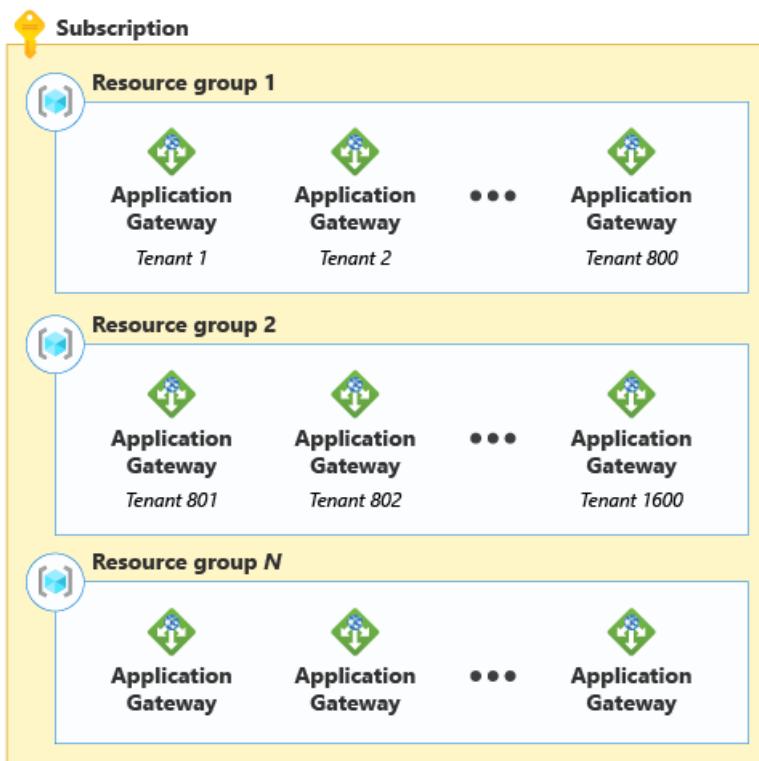


In the previous diagram, there are two application gateways. The first gateway is dedicated to customers 1 through 100, and the second is dedicated to customers 101 through 200.

Resource group and subscription limits

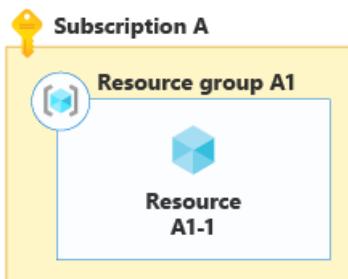
Whether you work with shared or dedicated resources, it's important to account for limits. Azure limits the number of resources that can be [deployed into a resource group](#) and [into an Azure subscription](#). As you approach these limits, you need to plan to scale across multiple resource groups or subscriptions.

For example, suppose you deploy a dedicated application gateway, for each of your customers, into a shared resource group. For some resources, [Azure supports deploying up to 800 resources of the same type](#) into a single resource group. So, when you reach this limit, you need to deploy any new application gateways into another resource group. In the following diagram, there are two resource groups. Each resource group contains 800 application gateways.

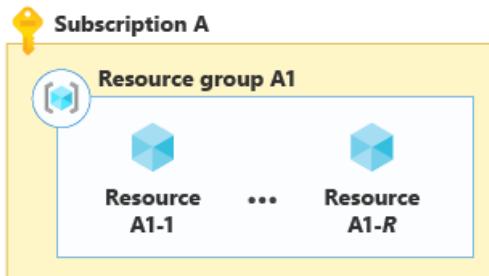


Bin pack tenants across resource groups and subscriptions

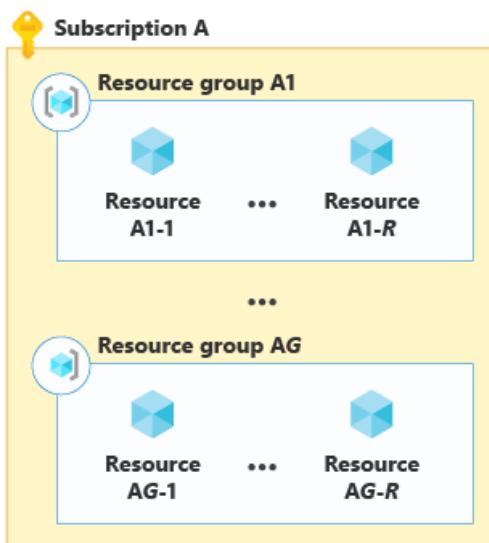
You can also apply the bin packing concept across resources, resource groups, and subscriptions. For example, when you have a small number of tenants you might be able to deploy a single resource and share it among all of your tenants. The following diagram shows bin packing into a single resource.



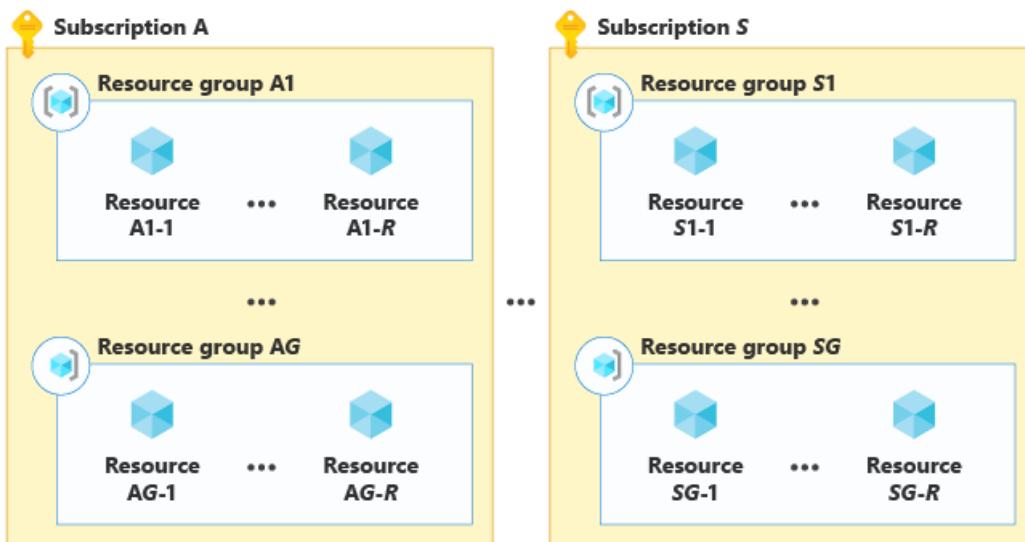
As you grow, you might approach the capacity limit for a single resource, and scale out to multiple (R) resources. The following diagram shows bin packing across multiple resources.



Over time, you might reach the limit of the number of resources in a single resource group, and you would then deploy multiple (R) resources into multiple (G) resource groups. The following diagram shows bin packing across multiple resources, in multiple resource groups.



And as you grow even larger, you can deploy across multiple (S) subscriptions, each containing multiple (G) resource groups with multiple (R) resources. The following diagram shows bin packing across multiple resources, in multiple resource groups and subscriptions.



By planning your scale-out strategy, you can scale to extremely large numbers of tenants and sustain a high level of load.

Tags

Resource tags enable you to add custom metadata to your Azure resources, which can be useful for management and tracking costs. For more details, see [Allocate costs by using resource tags](#).

Antipatterns to avoid

- **Not planning for scale.** Ensure you have a clear understanding of the limits of the resources you'll deploy, and which limits might become important, as your load or number of tenants increase. Plan how you'll deploy additional resources as you scale, and test the plan.
- **Not planning to bin pack.** Even if you don't need to grow immediately, plan to scale your Azure resources across multiple resources, resource groups, and subscriptions over time. Avoid making assumptions in your application code, like there being a single resource when you might need to scale to multiple resources in the future.
- **Scaling many individual resources.** If you have a complex resource topology, it can become difficult to scale individual components, one by one. It's often simpler to scale your solution as a unit, by following the [Deployment Stamps pattern](#).

- **Deploying isolated resources for each tenant, when not required.** In many solutions, it's more cost effective and efficient to deploy shared resources for multiple tenants.
- **Using separate Azure AD tenants.** In general, it's inadvisable to provision multiple Azure AD tenants. Managing resources across Azure AD tenants is complex. It's simpler to scale across subscriptions linked to a single Azure AD tenant.
- **Overarchitecting when you don't need to scale.** In some solutions, you know with certainty that you'll never grow beyond a certain level of scale. In these scenarios, there's no need to build complex scaling logic. However, if your organization plans to grow, then you will need to be prepared to scale—potentially at short notice.

Next steps

Review [Cost management and allocation](#) approaches.

Architectural approaches for governance and compliance in multitenant solutions

3/10/2022 • 8 minutes to read • [Edit Online](#)

As your use of Azure matures, it's important to consider the governance of your cloud resources. Governance includes how tenants' data is stored and managed, and how you organize your Azure resources. You might also need to follow regulatory or legal standards. This article provides information about how to consider governance and compliance in a multitenant solution. It also suggests some of the key Azure platform features that support these concerns.

Key considerations and requirements

Resource isolation

Ensure you configure your Azure resources to meet your tenants' isolation requirements. See [Azure resource organization in multitenant solutions](#) for guidance on isolating your Azure resources.

Data management

When you store data on behalf of your tenants, you might have requirements or obligations that you need to meet. From a tenant's perspective, they often expect ownership and control of their data. Consider how you isolate, store, access, and aggregate tenants' data. Uncover tenants' expectations and requirements that could affect how your solution works.

Isolation

Review the [Architectural approaches for storage and data in multitenant solutions](#) to understand how to isolate tenants' data. Consider whether tenants have requirements to use their own data encryption keys.

Whichever isolation approaches you implement, be prepared for tenants to request an audit of their data. It's a good practice to document all of the data stores in which tenants' data might be kept. Common data sources include the following:

- Databases and storage accounts deployed as part of your solution.
- Identity systems, which are often shared between tenants.
- Logs.
- Data warehouses.

Sovereignty

Understand whether there are any restrictions on the physical location for your tenants' data that's to be stored or processed. Your tenants might require you store their data in specific geographic locations. They might also require that you *don't* store their data in certain locations. Although these requirements are commonly based on legislation, they can also be based on cultural values and norms.

For more information about data residency and sovereignty, see the whitepaper [Enabling Data Residency and Data Protection in Microsoft Azure Regions](#).

Tenants' access to data that you store

Tenants sometimes request direct access to the data you store on their behalf. For example, they might want to ingest their data into their own data lake.

Plan how you'll respond to these requests. Consider whether any of the tenants' data is kept in shared data stores. If it is, plan how you'll avoid tenants accessing other tenants' data.

Avoid providing direct access to databases or storage accounts unless you designed for this requirement, such as by using the [Valet Key pattern](#). Consider creating an API or automated data export process for integration purposes.

Your access to tenants' data

Consider whether your tenants' requirements restrict the personnel who can work with their data or resources. For example, suppose you build a SaaS solution that's used by many different customers. A government agency might require that only citizens of their country are allowed to access the infrastructure and data for their solution. You might meet this requirement by using separate Azure resource groups, subscriptions, or management groups for sensitive customer workloads. You can apply tightly scoped Azure role-based access controls (RBAC) role assignments for specific groups of users to work with these resources.

Aggregation of data from multiple tenants

Consider whether you need to combine or aggregate data from multiple tenants. For example, do you analyze the aggregated data, or train machine learning models that could be applied to other tenants? Ensure your tenants understand the ways in which you use their data. Include any use of aggregated or anonymized data.

Compliance requirements

It's important that you understand whether you need to meet any compliance standards. Compliance requirements might be introduced in several situations, including:

- You, or any of your tenants, work within certain industries. For example, if any of your tenants work in the [healthcare industry](#), you might need to comply with the HIPAA standard.
- You, or any of your tenants, are located in geographic or geopolitical regions that require compliance with local laws. For example, if any of your tenants are located in Europe, you might need to comply with [General Data Protection Regulation \(GDPR\)](#).

IMPORTANT

Compliance is a shared responsibility between Microsoft, you, and your tenants.

Microsoft ensures that our services meet a specific set of compliance standards, and provides tools like [Microsoft Defender for Cloud](#) that help to verify your resources are configured according to those standards.

However, ultimately it is your responsibility to fully understand the compliance requirements that apply to your solution, and how to configure your Azure resources according to those standards. See [Azure compliance offerings](#) for more detail.

This article doesn't provide specific guidance about how to become compliant with any particular standards. Instead, it provides some general guidance around how to consider compliance and governance in a multitenant solution.

If different tenants need you to follow different compliance standards, plan to comply with the most stringent standard across your entire environment. It's easier to follow one strict standard than to follow different standards for different tenants.

Approaches and patterns to consider

Resource tags

Use [resource tags](#) to track the tenant identifier for tenant-specific resources, or the stamp identifier when you scale using the [Deployment Stamps pattern](#).

Access control

Use [Azure RBAC](#) to restrict access to the Azure resources that constitute the multitenant solution. Follow the [RBAC best practices](#), such as applying role assignments to groups instead of users. Scope your role assignments so they provide the minimum permissions necessary. Avoid long-standing access to resources by using just-in-time access and features like [Azure Active Directory Privileged Access Management](#).

Azure Resource Graph

[Azure Resource Graph](#) enables you to work with Azure resource metadata. By using Resource Graph, you can query across a large number of Azure resources, even if they're spread across multiple subscriptions. Resource Graph can query for the resources of a specific type, or to identify resources that have been configured in specific ways. It can also be used to track the history of a resource's configuration.

Resource Graph can be helpful to manage large Azure estates. For example, suppose you deploy tenant-specific Azure resources across multiple Azure subscriptions. By [applying tags to your resources](#), you can use the Resource Graph API to find resources that are used by specific tenants or deployment stamps.

Azure Purview

Consider using [Azure Purview](#) to track and classify the data that you store. When tenants request access to their data, you can easily determine the data sources that you should include.

Verify compliance with standards

Use tools like [Azure Policy](#), [Microsoft Defender for Cloud's regulatory compliance portal](#), and [Azure Advisor](#). These tools help you to configure your Azure resources to meet compliance requirements and to follow the recommended best practices.

Generate compliance documentation

Your tenants might require that you demonstrate your compliance with specific standards. Use the [Service Trust Portal](#) to generate compliance documentation that you can provide to your tenants or to third-party auditors.

Some multitenant solutions incorporate Microsoft 365 and use services like Microsoft OneDrive, Microsoft SharePoint, and Microsoft Exchange Online. The [Microsoft 365 Compliance Center](#) helps you understand how these services comply with regulatory standards.

Deployment Stamps pattern

Consider following the [Deployment Stamps pattern](#) when you need to comply with tenant-specific requirements.

For example, you might deploy stamps of your solution into multiple Azure regions. Then, you can assign new tenants to stamps, based on the regions that they need to have their data located in.

Similarly, a new tenant might introduce strict compliance requirements that you can't meet within your existing solution components. You can consider deploying a dedicated stamp for that tenant, and then configure it according to their requirements.

Antipatterns to avoid

- **Not understanding your tenants' compliance requirements.** It's important not to make assumptions about the compliance requirements that your tenants might impose. If you plan to grow your solution into new markets, be mindful of the regulatory environment that your tenants are likely to operate within.
- **Ignoring good practices.** If you don't have any immediate need to adhere to compliance standards, you should still follow good practices when you deploy your Azure resources. For example, isolate your resources, apply policies to verify resource configuration, and apply role assignments to groups instead of users. By following good practices, you make it simpler to follow compliance standards when you eventually need to do so.
- **Assuming there are no compliance requirements.** When you first launch a multitenant solution, you might not be aware of compliance requirements, or you might not need to follow any. As you grow, you'll likely need to provide evidence that you comply with various standards. Use [Microsoft Defender for Cloud](#) to monitor your compliance posture, even before you have an explicit requirement to do so.
- **Not planning for management.** As you deploy your Azure resources, consider how you plan to manage them. If you need to make bulk updates to resources, ensure you have an understanding of automation tools,

such as the Azure CLI, Azure PowerShell, Azure Resource Graph, and the Azure Resource Manager APIs.

- **Not using management groups.** Plan your subscription and management group hierarchy, including access control and Azure Policy resources at each scope. It can be difficult and disruptive to introduce or change these elements when your resources are used in a production environment.
- **Not planning your access control effectively.** Azure RBAC provides a high degree of control and flexibility in how you manage access to your resources. Ensure you use Azure AD groups to avoid assigning permissions to individual users. Assign roles at scopes that provide an appropriate balance between security and flexibility. Use built-in role definitions wherever possible, and assign roles that provide the minimum permissions required.
- **Not using Azure Policy.** It's important to use Azure Policy to govern your Azure environment. After you plan and deploy policies, ensure you monitor the policy compliance and carefully review any violations or exceptions.

Next steps

Review [approaches for cost management and allocation](#).

Architectural approaches for cost management and allocation in a multitenant solution

3/10/2022 • 7 minutes to read • [Edit Online](#)

Multitenant solutions often require special consideration when you measure and allocate costs, and when you optimize costs. On this page, we describe some key guidance for solution architects to consider about the measurement, allocation, and optimization of costs for a multitenant application.

Key considerations and requirements

Consider the requirements you have for measuring the consumption for your solution. This is discussed in more detail on [Measure the consumption of each tenant](#).

Purpose of measurement

It's important to decide what your goal is. The following are examples of goals:

- **Calculate an approximate cost of goods sold for each tenant.** For example, if you deploy a significant number of shared resources, you might only be interested in a rough approximation of the cost incurred for each tenant.
- **Calculate the exact cost incurred by each tenant.** For example, if you charge your tenants for the exact amount of consumption they incur, you need to have precise information about how much each tenant's resources cost.
- **Identify outlier tenants that cost significantly more than others.** For example, if you provide a [flat-rate pricing model](#), you might need to determine whether any tenants are consuming a disproportionate amount of your provisioned capacity, so that you can apply fair-use policies. In many situations, this use case doesn't require precise measurement of costs.
- **Reduce the overall Azure cost for your solution.** For example, you might want to look at the cost of every component, and then determine whether you have over-provisioned for the workload.

By understanding the goal of measuring the consumption by a tenant, you can determine whether the cost allocations need to be approximate or highly precise, which affects the specific tools you can use and the practices you can follow.

Shared components

You might be able to reduce the cost of a multitenant solution by moving tenants to shared infrastructure. However, you need to carefully consider the impact of sharing resources, such as whether your tenants will begin to experience the [Noisy Neighbor problem](#).

You also need to consider how you measure and allocate the costs of shared components. For example, you can evenly divide the cost between each of the tenants that use the shared component. Or, you can meter each tenant's usage to get a more precise measurement of their consumption of shared components.

Approaches and patterns to consider

Allocate costs by using resource tags

Azure enables you to [apply tags to your resources](#). A tag is a key-value pair. You use tags to add custom metadata. Tags are useful for many management operations, and they're also useful for analyzing the cost of your Azure consumption. After you apply tags, [you can determine costs associated with each tag](#).

The way you use tags in a multitenant solution is likely to be different, depending on your architecture.

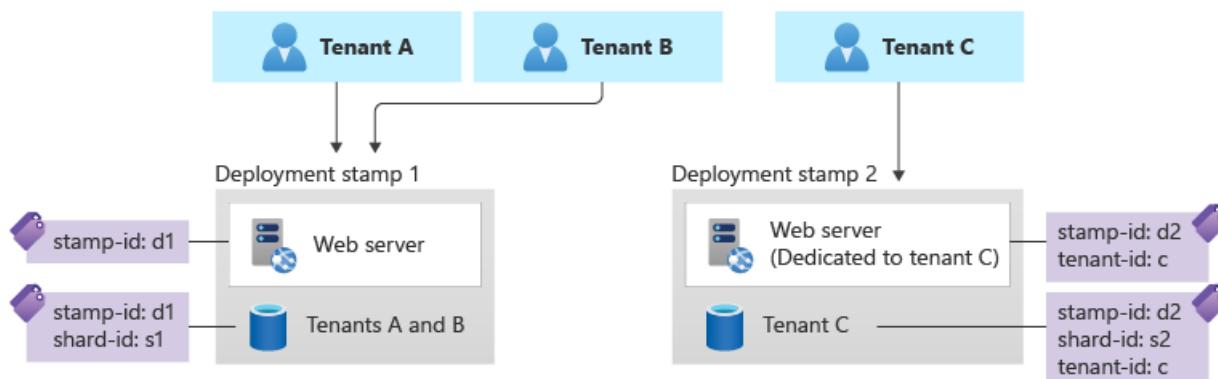
In some solutions, you might deploy dedicated resources for each tenant, such as if you deploy dedicated [Deployment Stamps](#) for each tenant. In these situations, it's clear that any Azure consumption for those resources should be allocated to that tenant, and so you can tag your Azure resources with the tenant ID.

In other situations, you might have sets of shared resources. For example, when you apply the [Sharding pattern](#), you might deploy multiple databases and spread your tenants across them. Consider tagging the resources with an identifier for the *group* of tenants. You might not be able to easily allocate costs to a single tenant, but you can at least narrow down the cost to a set of tenants, when you use this approach. You can also use the consumption information to help you rebalance tenants across the shards, if you notice that a specific shard is accruing higher costs than the others.

NOTE

There is a [limit to the number of tags](#) that can be applied to a resource. When you work with shared resources, it's best not to add a tag for every tenant that shares the resource. Instead, consider adding a tag with the shard ID or another way to identify the group of tenants.

Consider an example multitenant solution that's built using the [Deployment Stamps pattern](#). Each deployment stamp includes a shared web server and sharded databases. Tags can be applied to each of the Azure components, as shown in the following diagram.



The tagging strategy employed here is as follows:

- Every resource has a `stamp-id` tag.
- Every sharded database has a `shard-id` tag.
- Every resource dedicated to a specific tenant has a `tenant-id` tag.

With this tagging strategy, it's easy to filter the cost information to a single stamp. It's also easy to find the cost of the tenant-specific resources, such as the total cost of the database for tenant C. Shared components don't have a `tenant-id` tag, but the cost of the shared components for a stamp can be divided between the tenants who are assigned to use that stamp or shard.

Instrument your application

In situations where you don't have a direct relationship between an Azure resource and a tenant, consider instrumenting your application to collect telemetry.

Your application tier may already collect logs and metrics that are helpful to answer the following questions, for example:

- Approximately how many API requests are made per tenant?
- What times of the day are specific tenants busiest?
- How do tenant A's usage patterns compare to tenant B's usage patterns?

In Azure, these metrics are often captured by [Application Insights](#). By using [telemetry initializers](#), you can enrich the telemetry captured by Application Insights, to include a tenant identifier or other custom data.

However, Application Insights and other logging and monitoring solutions are not appropriate for precise cost measurement or for metering purposes. Application Insights is designed to [sample data](#), especially when your application has a high volume of requests. Sampling is designed to reduce the cost of monitoring your solution, because capturing every piece of telemetry can often become expensive.

If you need to track precise details about consumption or usage for billing purposes, you should instead build a custom pipeline to log the necessary data. You should then aggregate the data, based on your requirements. Azure services that can be helpful for this purpose include [Event Hubs](#), to capture large volumes of telemetry, and [Stream Analytics](#), to process it in real time.

Use Azure Reservations to reduce costs

[Azure Reservations](#) enable you to reduce your Azure costs by pre-committing to a certain level of spend.

Reservations apply to a number of Azure resource types.

Reservations can be used effectively in a multitenant solution. Note the following considerations:

- When you deploy a multitenant solution that includes shared resources, consider the baseline level of consumption that you need for the workload. You might consider a reservation for that baseline consumption, and then you'd pay standard rates for higher consumption during unpredictable peaks.
- When you deploy resources for each tenant, consider whether you can pre-commit to the resource consumption for a specific tenant, or across your portfolio of tenants.

Azure Reservations enables you to [scope your reservations](#) to apply to a resource group, a subscription, or a set of subscriptions. This means that you can take advantage of reservations, even if you shard your workload across multiple subscriptions.

Reservation scopes can also be helpful, when you have tenants with unpredictable workloads. For example, consider a solution in which tenant A only needs one instance of a specific resource, but tenants B and C each need two. Then tenant B becomes less busy, so you reduce the instance count, and tenant A gets busier, so you increase the instance count. Your reservations are applied to the tenants that need them.

Antipatterns to avoid

- **Not tracking costs at all.** It's important to have at least an approximate idea of the costs you're incurring, and how each tenant impacts the cost of delivering your solution. Otherwise, if your costs change over time, you have no baseline to compare against.
- **Making assumptions or guessing.** Ensure your cost measurement is based on real information. You might not need a high degree of precision, but even your estimates should be informed by real measurements.
- **Unnecessary precision.** You may not need to have a detailed accounting of every cost that's incurred for every tenant. Building unnecessarily precise cost measurement and optimization processes can be counterproductive, which adds engineering complexity and creates brittle processes.
- **Real-time measurement.** Most solutions don't need up-to-the-minute cost measurements. Because metering and consumption data can be complex to process, you should log the necessary data and then asynchronously aggregate and process the data later.
- **Using monitoring tools for billing.** As described in [Instrument your application](#), ensure you use tools that are designed for cost monitoring and metering. Application monitoring solutions are typically not good candidates for this type of data, especially when you need high precision.

Next steps

- Measure the consumption of each tenant

Architectural approaches for compute in multitenant solutions

3/10/2022 • 11 minutes to read • [Edit Online](#)

Most cloud-based solutions are composed of compute resources of some kind, such as web and application tiers, batch processors, scheduled jobs, and even specialized resources like GPUs and high-performance compute (HPC). Multitenant solutions often benefit from shared compute resources, because a higher density of tenants to infrastructure reduces the operational cost and management. You should consider the isolation requirements and the implications of shared infrastructure.

On this page, we provide guidance about the considerations and requirements that are essential for solution architects, when they're planning the compute services of a multitenant solution. This includes some common patterns for applying multitenancy to compute services, along with some antipatterns to avoid.

Key considerations and requirements

Multitenancy, and the isolation model you select, impacts the scaling, performance, state management, and security of your compute resources. In this section, we review some of the key decisions you must make when you plan a multitenant compute solution.

Scale

Systems need to perform adequately under changing demand. As the number of tenants and the amount of traffic increase, you might need to increase the capacity of your resources, to keep up with the growing number of tenants and to maintain an acceptable performance rate. Similarly, when the number of active users or the amount of traffic decrease, you should automatically reduce the compute capacity to reduce costs, but you should reduce the capacity with minimal impact to users.

If you deploy dedicated resources for each tenant, you have the flexibility to scale each tenant's resources independently. In a solution where compute resources are shared between multiple tenants, if you scale those resources, then all of those tenants can make use of the new scale. However, they also will all suffer when the scale is insufficient to handle their overall load. For more information, see the [Noisy Neighbor problem](#).

When you build cloud solutions, you can choose whether to [scale horizontally or vertically](#). In a multitenant solution with a growing number of tenants, scaling horizontally typically provides you with greater flexibility and a higher overall scale ceiling.

Performance problems often remain undetected until an application is under load. You can use a fully managed service, such as [Azure Load Testing Preview](#), to learn how your application behaves under stress.

Scale triggers

Whichever approach you use to scale, you typically need to plan the triggers that cause your components to scale. When you have shared components, consider the workload patterns of every tenant who uses the resources, in order to ensure you provision capacity can meet the total required capacity, and to minimize the chance of a tenant experiencing the [Noisy Neighbor problem](#). You might also be able to plan your scaling capacity, based on the number of tenants. For example, if you measure the resources that you use to service 100 tenants, then as you onboard more tenants, you can plan to scale such that your resources double for every additional 100 tenants.

State

Compute resources can be *stateless*, or they can be *stateful*. Stateless components don't maintain any data between requests. From a scalability perspective, stateless components are often easy to scale out because you

can quickly add new workers, instances, or nodes, and they can immediately start to process requests. If your architecture allows for it, you can also repurpose instances that are assigned to one tenant and allocate them to another tenant.

Stateful resources can be further subdivided, based on the type of state they maintain. *Persistent state* is data that needs to be permanently stored. In cloud solutions, you should avoid storing a persistent state in your compute tier. Instead, use storage services like databases or storage accounts. *Transient state* is data that is stored temporarily, and it includes read-only in-memory caches, and the storage of temporary files on local disks.

Transient state is often useful to improve the performance of your application tier, by reducing the number of requests to backend storage services. For example, when you use an in-memory cache, you might be able to serve read requests, without connecting to a database, and without performing an intensive query that you recently performed when you served another request.

In latency-sensitive applications, the cost of cache hydration can become significant. A multitenant solution can exacerbate this issue, if each tenant requires different data to be cached. To mitigate this issue, some solutions use *session affinity* to ensure that all requests for a specific user or tenant are processed by the same compute worker node. Although session affinity can improve the ability of the application tier to use its cache effectively, it also makes it harder to scale and to balance the traffic load across workers. This tradeoff needs to be carefully considered. For many applications, session affinity is not required.

It's also possible to store data in external caches, such as Azure Cache for Redis. External caches are optimized for low-latency data retrieval, while keeping the state isolated from the compute resources, so they can be scaled and managed separately. In many solutions, external caches enable you to improve application performance, while you keep the compute tier stateless.

IMPORTANT

Avoid leaking data between tenants, whenever you use in-memory caches or other components that maintain state. For example, consider prepending a tenant identifier to all cache keys, to ensure that data is separated for each tenant.

Isolation

When you design a multitenant compute tier, you often have many options to consider for the level of isolation between tenants, including deploying [shared compute resources](#), to be used by all tenants, [dedicated compute resources](#) for each tenant, or [something in between these extremes](#). Each option comes with tradeoffs. To help you decide which option suits your solution best, consider your requirements for isolation.

You might be concerned with the logical isolation of tenants, and how to separate the management responsibilities or policies that are applied to each tenant. Alternatively, you might need to deploy distinct resource configurations for specific tenants, such as deploying a specific virtual machine SKU to suit a tenant's workload.

Whichever isolation model you select, ensure you verify your tenant data remains appropriately isolated even when components are unavailable or malfunctioning. Consider using [Azure Chaos Studio](#) as part of your regular automated testing process to deliberately introduce faults that simulate real-world outages and verify that your solution doesn't leak data between tenants and is functioning properly even under pressure.

Approaches and patterns to consider

Autoscale

Azure compute services provide different capabilities to scale your workloads. [Many compute services support autoscaling](#), which requires you to consider when you should scale, and your minimum and maximum levels of scale. The specific options available for scaling depend on the compute services you use. See the following

example services:

- **Azure App Service**: [Specify autoscale rules](#) that scale your infrastructure, based on your requirements.
- **Azure Functions**: Select from [multiple scale options](#), including an event-driven scaling model that automatically scales, based on the work that your functions perform.
- **Azure Container Apps**: Use [event-driven autoscaling](#) to scale your application, based on the work it performs and on its current load.
- **Azure Kubernetes Service (AKS)**: To keep up with your application's demands, you might need to [adjust the number of nodes that run your workloads](#). Additionally, to rapidly scale application workloads in an AKS cluster, you can use [virtual nodes](#).
- **Virtual machines**: A virtual machine scale set can [automatically increase or decrease the number of VM instances](#) that run your application.

Deployment Stamps pattern

For more information about how the [Deployment Stamps pattern](#) can be used to support a multitenant solution, see [Overview](#).

Compute Resource Consolidation pattern

The [Compute Resource Consolidation pattern](#) helps you achieve a higher density of tenants to compute infrastructure, by sharing the underlying compute resources. By sharing compute resources, you are often able to reduce the direct cost of those resources. Additionally, your management costs are often lower because there are fewer components to manage.

However, compute resource consolidation increases the likelihood of the [Noisy Neighbor problem](#). Any tenant's workload might consume a disproportionate amount of the compute capacity that's available. You can often mitigate this risk by ensuring you scale your solution appropriately, and by applying controls like quotas and API limits, to avoid tenants that consume more than their fair share of the capacity.

This pattern is achieved in different ways, depending on the compute service you use. See the following example services:

- **Azure App Service and Azure Functions**: Deploy shared App Service plans, which represent the hosting server infrastructure.
- **Azure Container Apps**: Deploy shared environments.
- **Azure Kubernetes Service (AKS)**: Deploy shared pods, with a multitenancy-aware application.
- **Virtual machines**: Deploy a single set of virtual machines for all tenants to use.

Dedicated compute resources per tenant

You can also deploy dedicated compute resources for every tenant. Dedicated resources typically come with a higher cost, because you have a lower density of tenants to resources. However, dedicated resources mitigate the risk of the [Noisy Neighbor problem](#), by ensuring that the compute resources for every tenant are isolated from the others. It also enables you to deploy a distinct configuration for each tenant's resources, based on their requirements.

Depending on the Azure compute services you use, you need to deploy different dedicated resources, as follows:

- **Azure App Service and Azure Functions**: Deploy separate App Service plans for each tenant.
- **Azure Container Apps**: Deploy separate environments for each tenant.
- **Azure Kubernetes Service (AKS)**: Deploy dedicated clusters for each tenant.
- **Virtual machines**: Deploy dedicated virtual machines for each tenant.

Semi-isolated compute resources

Semi-isolated approaches require you to deploy aspects of the solution in an isolated configuration, while you share the other components.

When you work with App Service and Azure Functions, you can deploy distinct applications for each tenant, and you can host the applications on shared App Service plans. This approach reduces the cost of your compute tier, because App Service plans represent the unit of billing. It also enables you to apply distinct configuration and policies to each application. However, this approach introduces the risk of the [Noisy Neighbor problem](#).

Azure Container Apps enables you to deploy multiple applications to a shared environment, and then to use Dapr and other tools to configure each application separately.

Azure Kubernetes Service (AKS), and Kubernetes more broadly, provide a variety of options for multitenancy, including the following:

- Tenant-specific namespaces, for logical isolation of tenant-specific resources, which are deployed to shared clusters and node pools.
- Tenant-specific nodes or node pools on a shared cluster.
- Tenant-specific pods that might use the same node pool.

AKS also enables you to apply pod-level governance to mitigate the [Noisy Neighbor problem](#). For more information, see [Best practices for application developers to manage resources in Azure Kubernetes Service \(AKS\)](#).

It's also important to be aware of shared components in a Kubernetes cluster, and how these components might be affected by multitenancy. For example, the Kubernetes API server is a shared service that is used throughout the entire cluster. Even if you provide tenant-specific node pools to isolate the tenants' application workloads, the API server might experience contention from a large number of requests across the tenants.

Antipatterns to avoid

Noisy Neighbor antipattern

Whenever you deploy components that are shared between tenants, the [Noisy Neighbor problem](#) is a potential risk. Ensure you include resource governance and monitoring to mitigate the risk of a tenant's compute workload being affected by the activity of other tenants.

Cross-tenant data leakage

Compute tiers can be subject to cross-tenant data leakage, if they are not properly handled. This isn't generally something you need to consider when you're using a multitenant service on Azure, because Microsoft provides protections at the platform layer. However, when you develop your own multitenant application, consider whether any shared resources (such as local disk caches, RAM, and external caches) might contain data that another tenant can inadvertently view or modify.

Busy Front End antipattern

To avoid the [Busy Front End antipattern](#), avoid your front end tier doing a lot of the work that could be handled by other components or tiers of your architecture. This antipattern is particularly important when you create shared front-ends for a multitenant solution, because a busy front end will degrade the experience for all tenants.

Instead, consider using asynchronous processing by making use of queues or other messaging services. This approach also enables you to apply *quality of service* (QoS) controls for different tenants, based on their requirements. For example, all tenants might share a common front end tier, but tenants who [pay for a higher service level](#) might have a higher set of dedicated resources to process the work from their queue messages.

Inelastic or insufficient scaling

Multitenant solutions are often subject to bursty scale patterns. Shared components are particularly susceptible to this issue, because the scope for burst is higher, and the impact is greater when you have more tenants with distinct usage patterns.

Ensure you make good use of the elasticity and scale of the cloud. Consider whether you should use [horizontal or vertical scaling](#), and use autoscaling to automatically handle spikes in load. Test your solution to understand how it behaves under different levels of load. Ensure you include the load volumes that are expected in production, and your expected growth. You can use a fully managed service, such as [Azure Load Testing Preview](#), to learn how your application behaves under stress.

No Caching antipattern

The [No Caching antipattern](#) is when the performance of your solution suffers because the application tier repeatedly requests or recomputes information that could be reused across requests. If you have data that can be shared, either among tenants or among users within a single tenant, it's likely worth caching it to reduce the load on your backend/database tier.

Unnecessary statefulness

The corollary to the No Caching antipattern is that you also should avoid storing unnecessary state in your compute tier. Be explicit about where you maintain state and why. Stateful front-end or application tiers can reduce your ability to scale. Stateful compute tiers typically also require session affinity, which can reduce your ability to effectively load balance traffic, across workers or nodes.

Consider the tradeoffs for each piece of state you maintain in your compute tier, and whether it impacts your ability to scale or to grow as your tenants' workload patterns change. You can also store state in an external cache, such as Azure Cache for Redis.

Next steps

Consider [architectural approaches for storage and data](#).

Architectural approaches for networking in multitenant solutions

3/10/2022 • 13 minutes to read • [Edit Online](#)

All solutions deployed to Azure require networking of some kind. Depending on your solution design and the workload, the ways in which you interact with Azure's networking services might be different. In this article, we provide considerations and guidance for the networking aspects of multitenant solutions on Azure. We include information about the lower-level networking components, like virtual networks, through to higher-level and application-tier approaches.

NOTE

Azure itself is a multitenant environment, and Azure's network components are designed for multitenancy. Although it's not required to understand the details in order to design your own solution, you can [learn more about how Azure isolates your virtual network traffic from other customers' traffic](#).

Key considerations and requirements

Infrastructure and platform services

The concerns you have for your networking components will differ, depending on the category of services you use.

Infrastructure services

Whenever you use infrastructure services, like virtual machines or Azure Kubernetes Service, you need to consider and design the virtual networks, or VNets, that underpin your services' connectivity. You also need to consider the other layers of security and isolation that you need to incorporate in your design. [Avoid relying exclusively on network-layer controls](#).

Platform services

If you use Azure's platform services, like App Service, Azure Cosmos DB, or Azure SQL Database, then the specific architecture you use will determine the networking services you require.

If you need to isolate your platform services from the internet, you need to use a VNet. Depending on the specific services you use, you might work with [private endpoints](#) or VNet-integrated resources, like [Application Gateway](#). However, you might also choose to make your platform services available through their public IP addresses, and use the services' own protections like firewalls and identity controls. In these situations, you might not need a VNet.

The decision of whether to use VNets for platform services is based on many requirements, including the following factors:

- **Compliance requirements:** You might need to meet a specific compliance standard. Some standards require your Azure environment to be configured in specific ways.
- **Your tenants' requirements:** Even if your own organization doesn't have specific requirements for network-layer isolation or controls, your tenants might. Ensure you have a clear understanding of how your tenants will access your solution and whether they have any assumptions about its network design.
- **Complexity:** It can be more complex to understand and work with virtual networks. Ensure your team has a clear understanding of the principles involved, or you're likely to deploy an insecure environment.

Ensure that you understand the [implications of using private networking](#).

Sizing subnets

When you need to deploy a VNet, it's important to carefully consider the sizing and address space of the entire VNet and of the subnets within the VNet.

Ensure you have a clear understanding of how you will deploy your Azure resources into VNets, and the number of IP addresses each resource consumes. If you deploy tenant-specific compute nodes, database servers, or other resources, ensure you create subnets that are large enough for your expected tenant growth and [horizontal autoscaling of resources](#).

Similarly, when you work with managed services, it's important that you understand how IP addresses are consumed. For example, when you use Azure Kubernetes Service with [Azure Container Networking Interface \(Azure CNI\)](#), the number of IP addresses consumed from a subnet will be based on factors like the number of nodes, how you scale horizontally, and the service deployment process that you use. When you use Azure App Service and Azure Functions with VNet integration, [the number of IP addresses consumed is based on the number of plan instances](#).

[Review the subnet segmentation guidance](#) when planning your subnets.

Public or private access

Consider whether your tenants will access your services through the internet or through private IP addresses.

For internet-based (public) access, you can use firewall rules, IP address allowlisting and denylisting, shared secrets and keys, and identity-based controls to secure your service.

If you need to enable connectivity to your service by using private IP addresses, consider using [Azure Private Link Service](#) or [cross-tenant virtual network peering](#). For some limited scenarios, you might also consider using Azure ExpressRoute or Azure VPN Gateway to enable private access to your solution. We only advise that you use this approach, when you have dedicated networks for each tenant, and a small number of tenants.

Access to tenants' endpoints

Consider whether you need to send data to endpoints within the tenants' networks, either within or outside of Azure. For example, will you need to invoke a webhook that's provided by a customer, or do you need to send real-time messages to a tenant?

If you do need to send data to tenants' endpoints, consider the following common approaches:

- Initiate connections from your solution to tenants' endpoints through the internet. Consider whether the connections must originate from a [static IP address](#). Depending on the Azure services you use, you might need to deploy a [NAT Gateway](#), firewall, or load balancer.
- Deploy an [agent](#) to enable connectivity between your Azure-hosted services and your customers' networks, regardless of where they are located.
- For one-way messaging, consider using a service like [Azure Event Grid](#), with or without [event domains](#).

Approaches and patterns to consider

In this section, we describe some of the key networking approaches that you can consider in a multitenant solution. We begin by describing the lower-level approaches for core networking components, and then follow with the approaches that you can consider for HTTP and other application-layer concerns.

Tenant-specific VNets with service provider-selected IP addresses

In some situations, you need to run dedicated VNet-connected resources in Azure on a tenant's behalf. For example, you might run a virtual machine for each tenant, or you might need to use private endpoints to access tenant-specific databases.

Consider deploying a VNet for each tenant, by using an IP address space that you control. This approach enables you to peer the VNets together for your own purposes, such as if you need to establish a [hub and spoke](#)

[topology](#) to centrally control traffic ingress and egress.

However, service provider-selected IP addresses aren't appropriate if tenants need to connect directly to the VNet you created, such as by using VNet peering. It's likely that the address space you select will be incompatible with their own address spaces.

Tenant-specific VNets with tenant-selected IP addresses

If tenants need to peer their own VNets with the VNet you manage on their behalf, consider deploying tenant-specific VNets with an IP address space that the tenant selects. This system enables each tenant to ensure that the IP address ranges in the VNet you control do not overlap with their own VNets and are compatible for peering.

However, this approach means it's unlikely that you can peer your tenants' VNets together or adopt a [hub and spoke topology](#), because there are likely to be overlapping IP address ranges among VNets of different tenants.

Hub and spoke topology

The [hub and spoke VNet topology](#) enables you to peer a centralized *hub* VNet with multiple *spoke* VNets. You can centrally control the traffic ingress and egress for your VNets, and control whether the resources in each spoke's VNet can communicate with each other. Each spoke VNet can also access shared components, like Azure Firewall, and it might be able to use services like Azure DDoS Protection.

When you use a hub and spoke topology, ensure you plan around limits, [such as the maximum number of peered VNets](#), and ensure that you don't use overlapping address spaces for each tenant's VNet.

The hub and spoke topology can be useful when you deploy tenant-specific VNets with IP addresses that you select. Each tenant's VNet becomes a spoke, and can share your common resources in the hub VNet. You can also use the hub and spoke topology when you scale shared resources across multiple VNets for scale purposes, or when you use the [Deployment Stamps pattern](#).

TIP

If your solution runs across multiple geographic regions, it's usually a good practice to deploy separate hubs and hub resources in each region. While this practice incurs a higher resource cost, it avoids traffic going through multiple Azure regions unnecessarily, which can increase the latency of requests and incur global peering charges.

Static IP addresses

Consider whether your tenants need your service to use static public IP addresses for inbound traffic, outbound traffic, or both. Different Azure services enable static IP addresses in different ways.

When you work with virtual machines and other infrastructure components, consider using a load balancer or firewall for both inbound and outbound static IP addressing. Consider using NAT Gateway to control the IP address of outbound traffic.

When you work with platform services, the specific service you use determines whether and how you can control IP addresses. You might need to configure the resource in a specific way, such as by deploying the resource into a VNet and by using a NAT Gateway or firewall. Or, you can request the current set of IP addresses that the service uses for outbound traffic. For example, [App Service provides an API and web interface to obtain the current outbound IP addresses for your application](#).

Agents

If you need to enable your tenants to receive messages that are initiated by your solution, or if you need to access data that exists in tenants' own networks, then consider providing an agent (sometimes called an *on-premises gateway*) that they can deploy within their network. This approach can work whether your tenants' networks are in Azure, in another cloud provider, or on premises.

The agent initiates an outbound connection to an endpoint that you specify and control, and either keeps long-

running connections alive or polls intermittently. Consider using [Azure Relay](#) to establish and manage connections from your agent to your service. When the agent establishes the connection, it authenticates and includes some information about the tenant identifier so that your service can map the connection to the correct tenant.

Agents typically simplify the security configuration for your tenants. It can be complex and risky to open inbound ports, especially in an on-premises environment. An agent avoids the need for tenants to take this risk.

Examples of Microsoft services that provide agents for connectivity to tenants' networks include:

- [Azure Data Factory's self-hosted integration runtime](#).
- [Azure App Service Hybrid Connection](#).
- Microsoft on-premises data gateway, which is used for [Azure Logic Apps](#), [Power BI](#), and other services.

Azure Private Link service

[Azure Private Link service](#) provides private connectivity from a tenant's Azure environment to your solution. Tenants can also use Private Link service with their own VNet, to access your service from an on-premises environment.

Tenants can deploy a private endpoint within their VNet and configure it to your Private Link service instance. Azure securely routes the traffic to the service. Azure Private Link service is used by many large SaaS providers, including [Snowflake](#), [Confluent Cloud](#), and [MongoDB Atlas](#).

[Private endpoints typically require approval](#), when the destination resource is in a different Azure subscription to the resource. You can [automate the approval process](#) within your solution by using Azure PowerShell, the Azure CLI, and the Azure Resource Manager API.

Domain names, subdomains, and TLS

When you work with domain names and transport-layer security (TLS) in a multitenant solution, there are a number of considerations. [Review the considerations for multitenancy and domain names](#).

Gateway Routing and Gateway Offloading patterns

The [Gateway Routing pattern](#) and the [Gateway Offloading pattern](#) involve deploying a layer 7 reverse proxy or *gateway*. Gateways are useful to provide core services for a multitenant application, including the following capabilities:

- Routing requests to tenant-specific backends or deployment stamps.
- Handling tenant-specific domain names and TLS certificates.
- Inspecting requests for security threats, by using a [web application firewall \(WAF\)](#).
- Caching responses to improve performance.

Azure provides several services that can be used to achieve some or all of these goals, including Azure Front Door, Azure Application Gateway, and Azure API Management. You can also deploy your own custom solution, by using software like NGINX or HAProxy.

If you plan to deploy a gateway for your solution, a good practice is to first build a complete prototype that includes all of the features you need, and to verify that your application components continue to function as you expect. You should also understand how the gateway component will scale to support your traffic and tenant growth.

Static Content Hosting pattern

The [Static Content Hosting pattern](#) involves serving web content from a cloud-native storage service, and using a content delivery network (CDN) to cache the content.

You can use [Azure Front Door](#) or another CDN for your solution's static components, such as single-page JavaScript applications, and for static content like image files and documents.

Depending on how your solution is designed, you might also be able to cache tenant-specific files or data within a CDN, such as JSON-formatted API responses. This practice can help you improve the performance and scalability of your solution, but it's important to consider whether tenant-specific data is isolated sufficiently to avoid leaking data across tenants. Consider how you plan to purge tenant-specific content from your cache, such as when data is updated or a new application version is deployed. By including the tenant identifier in the URL path, you can control whether you purge a specific file, all the files that relate to a specific tenant, or all the files for all the tenants.

Antipatterns to avoid

Failing to plan for VNet connectivity

By deploying resources into VNets, you have a great deal of control over how traffic flows through your solution's components. However, VNet integration also introduces additional complexity, cost, and other factors that you need to consider. This effect is especially true with platform at a service (PaaS) components.

It's important to test and plan your network strategy, so that you uncover any issues before you implement it in a production environment.

Not planning for limits

Azure enforces a number of limits that affect networking resources. These limits include [Azure resource limits](#) and fundamental protocol and platform limits. For example, when you build a high-scale multitenant solution on platform services, such as Azure App Service and Azure Functions, you might need to consider the [number of TCP connections and SNAT ports](#). When you work with virtual machines and load balancers, you also need to consider limitations for [outbound rules](#) and for [SNAT ports](#).

Small subnets

It's important to consider the size of each subnet to allow for the number of resources or instances of resources that you will deploy. When you work with platform as a service (PaaS) resources, ensure you understand how your resource's configuration and scale will affect the number of IP addresses that are required in its subnet.

Improper network segmentation

If your solution requires virtual networks, consider how you configure [network segmentation](#) to enable you to control inbound and outbound (north-south) traffic flows and the flows within your solution (east-west). Decide whether tenants should have their own VNets, or if you will deploy shared resources in shared VNets. Changing the approach can be difficult, so ensure you consider all of your requirements, and then select an approach that will work for your future growth targets.

Relying only on network-layer security controls

In modern networks, it's important to combine network-layer security with other security controls, and you should not rely only on firewalls or network segmentation. This is sometimes called *zero-trust networking*. Use identity-based controls to verify the client, caller, or user, at every layer of your solution. Consider using services that enable you to add additional layers of protection. The options you have available depend on the Azure services that you use. In AKS, consider using a service mesh for mutual TLS authentication, and [network policies](#) to control east-west traffic. In App Service, consider using the [built-in support for authentication and authorization](#) and [access restrictions](#).

Rewriting host headers without testing

When you use the [Gateway Offloading pattern](#), you might consider rewriting the `Host` header of HTTP requests. This practice can simplify the configuration of your backend web application service by offloading the custom domain and TLS management to the gateway.

However, `Host` header rewrites can cause problems for some backend services. If your application issues HTTP redirects or cookies, the mismatch in host names can break the application's functionality. In particular, this issue can arise when you use backend services that are themselves multitenant, like Azure App Service, Azure

Functions, and Azure Spring Cloud.

Ensure you test your application's behavior with the gateway configuration that you plan to use.

Next steps

Review [considerations when using domain names in a multitenant solution](#).

Architectural approaches for storage and data in multitenant solutions

3/10/2022 • 14 minutes to read • [Edit Online](#)

When planning multitenant storage or data components, you need to decide on an approach for sharing or isolating your tenants' data. Data is often considered the most valuable part of a solution, since it represents your or your customers' valuable business information. So, it's important to carefully plan the approach you use to manage data in a multitenant environment. On this page, we provide guidance about the key considerations and requirements that are essential for solution architects when deciding on an approach to store data in a multitenant system. We then suggest some common patterns for applying multitenancy to storage and data services, and some antipatterns to avoid. Finally, we provide targeted guidance for some specific situations.

Key considerations and requirements

It's important to consider the approaches you use for storage and data services from a number of perspectives, which approximately align to the pillars of the [Azure Well-Architected Framework](#).

Scale

When working with services that store your data, you should consider the number of tenants you have, and the volume of data you store. If you have a small number of tenants (such as five or less), and you're storing small amounts of data for each tenant, then it's likely to be a wasted effort to plan a highly scalable data storage approach, or to build a fully automated approach to manage your data resources. As you grow, you will increasingly benefit from having a clear strategy to scale your data and storage resources, and to apply automation to their management. When you have 50 tenants or more, or if you plan to reach that level of scale, then it's especially important to design your data and storage approach, with scale as a key consideration.

Consider the extent to which you plan to scale, and clearly plan your data storage architectural approach to meet that level of scale.

Performance predictability

Multitenant data and storage services are particularly susceptible to the [Noisy Neighbor problem](#). It's important to consider whether your tenants could affect each other's performance. For example, do your tenants have overlapping peaks in their usage patterns over time? Do all of your customers use your solution at the same time each day, or are requests distributed evenly? Those factors will impact the level of isolation you need to design for, the amount of resources you need to provision, and the degree to which resources can be shared between tenants.

It's important to consider [Azure's resource and request quotas](#) as part of this decision. For example, suppose you deploy a single storage account to contain all of your tenants' data. If you exceed a specific number of storage operations per second, Azure Storage will reject your application's requests, and all of your tenants will be impacted. This is called *throttling* behavior. It's important that you monitor for throttled requests. See [Retry guidance for Azure services](#) for further information.

Data isolation

When designing a solution that contains multitenant data services, there are usually different options and levels of data isolation, each with their own benefits and tradeoffs. For example:

- When using Azure Cosmos DB, you can deploy separate containers for each tenant, and you can share databases and accounts between multiple tenants. Alternatively, you might consider deploying different databases or even accounts for each tenant, depending on the level of isolation required.

- When using Azure Storage for blob data, you can deploy separate blob containers for each tenant, or you can deploy separate storage accounts.
- When using Azure SQL, you can use separate tables in shared databases, or you can deploy separate databases or servers for each tenant.
- In all Azure services, you can consider deploying resources within a single shared Azure subscription, or you can use multiple Azure subscriptions--perhaps even one per tenant.

There is no single solution that works for every situation. The option you choose depends on a number of factors and the requirements of your tenants. For example, if your tenants need to meet specific compliance or regulatory standards, you might need to apply a higher level of isolation. Similarly, you might have commercial requirements to physically isolate your customers' data, or you might need to enforce isolation to avoid the [Noisy Neighbor problem](#). Additionally, if tenants need to use their own encryption keys, they have individual backup and restore policies, or they need to have their data stored in different geographical locations, you might need to isolate them from other tenants, or group them with tenants that have similar policies.

Complexity of implementation

It's important to consider the complexity of your implementation. It's good practice to keep your architecture as simple as possible, while still meeting your requirements. Avoid committing to an architecture that will become increasingly complex as you scale, or an architecture that you don't have the resources or expertise to develop and maintain.

Similarly, if your solution doesn't need to scale to a large number of tenants, or if you don't have concerns around performance or data isolation, then it's better to keep your solution simple and avoid adding unnecessary complexity.

A particular concern for multitenant data solutions is the level of customization you support. For example, can a tenant extend your data model or apply custom data rules? Ensure you design for this upfront. Avoid forking or providing custom infrastructure for individual tenants, since this inhibits your ability to scale, to test your solution, and to deploy updates. Instead, consider using [feature flags](#) and other forms of tenant configuration.

Complexity of management and operations

Consider how you plan to operate your solution, and how your multitenancy approach affects your operations and processes. For example:

- Consider cross-tenant management operations, such as regular maintenance activities. If you use multiple accounts, servers, or databases, how will you initiate and monitor the operations for each tenant?
- If you monitor or meter your tenants, consider how your solution reports metrics, and whether they can be easily linked to the tenant that triggered the request.
- Reporting data from across isolated tenants may require that each tenant publishes data to a centralized data warehouse, rather than running queries on each database individually and then aggregating the results.
- If you use a database that enforces a schema, plan how you will deploy schema updates across your estate. Consider how your application knows which schema version to use for a specific tenant's database queries.
- Consider your tenants' high availability requirements (for example, uptime service level agreements, or SLAs) and disaster recovery requirements (for example, recovery time objectives, or RTOs, and recovery point objectives, or RPOs). If tenants have different expectations, will you be able to meet each tenant's requirements?
- How will you migrate tenants if they need to move to a different type of service, a different deployment, or another region?

Cost

Generally, the higher the density of tenants to your deployment infrastructure, the lower the cost to provision that infrastructure. However, shared infrastructure increases the likelihood of issues like the [Noisy Neighbor problem](#), so consider the tradeoffs carefully.

Approaches and patterns to consider

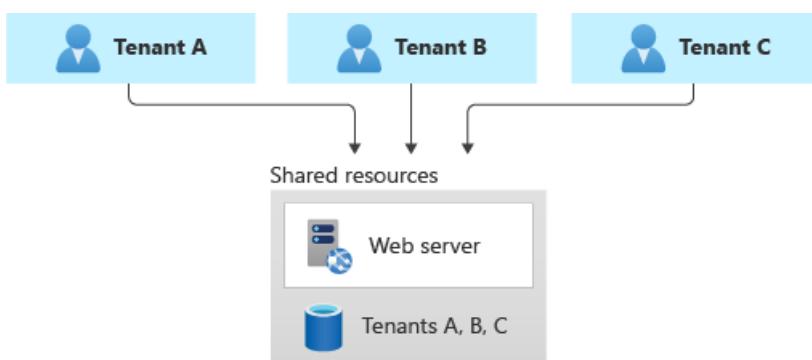
Several design patterns from the Azure Architecture Center are of relevance to multitenant storage and data services. You might choose to follow one pattern consistently. Or, you could consider mixing and matching patterns. For example, you might use a multitenant database for most of your tenants, but deploy single-tenant stamps for tenants who pay more or who have unusual requirements. Similarly, it's often a good practice to scale by using deployment stamps, even when you use a multitenant database or sharded databases within a stamp.

Deployment Stamps pattern

For more information about how the [Deployment Stamps pattern](#) can be used to support a multitenant solution, see [Overview](#).

Shared multitenant databases and file stores

You might consider deploying a shared multitenant database, storage account, or file share, and sharing it across all of your tenants.



This approach provides the highest density of tenants to infrastructure, so it tends to come at the lowest cost of any approach. It also often reduces the management overhead, since there's a single database or resource to manage, back up, and secure.

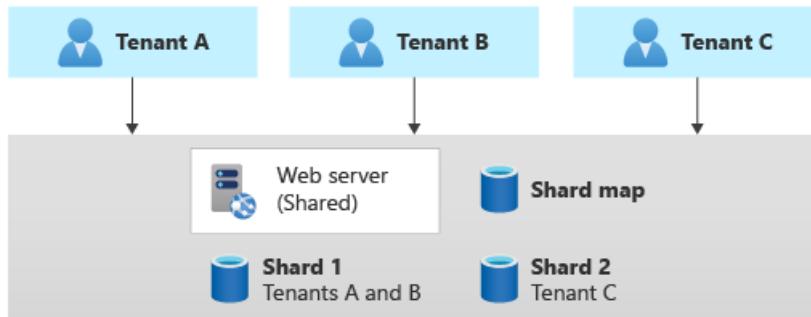
However, when you work with shared infrastructure, there are several caveats to consider:

- When you rely on a single resource, consider the supported scale and limits of that resource. For example, the maximum size of one database or file store, or the maximum throughput limits, will eventually become a hard blocker, if your architecture relies on a single database. Carefully consider the maximum scale you need to achieve, and compare it to your current and future limits, before you select this pattern.
- The [Noisy Neighbor problem](#) might become a factor, especially if you have tenants that are particularly busy or generate higher workloads than others. Considering applying the [Throttling pattern](#) or the [Rate Limiting pattern](#) to mitigate these effects.
- You might have difficulty monitoring the activity and [measuring the consumption](#) for a single tenant. Some services, such as Azure Cosmos DB, provide reporting on resource usage for each request, so this information can be tracked to measure the consumption for each tenant. Other services don't provide the same level of detail. For example, the Azure Files metrics for file capacity are available per file share dimension, only when you use premium shares. However, the standard tier provides the metrics only at the storage account level.
- Tenants may have different requirements for security, backup, availability, or storage location. If these don't match your single resource's configuration, you might not be able to accommodate them.
- When working with a relational database, or another situation where the schema of the data is important, then tenant-level schema customization is difficult.

Sharding pattern

The [Sharding pattern](#) involves deploying multiple separate databases, called *shards*, that contain one or more tenants' data. Unlike deployment stamps, shards don't imply that the entire infrastructure is duplicated. You

might shard databases without also duplicating or sharding other infrastructure in your solution.



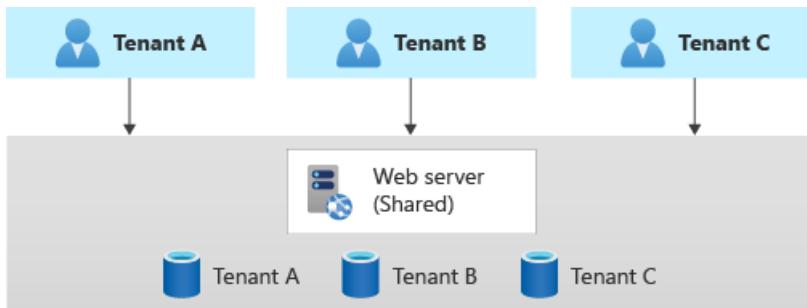
Sharding is closely related to *partitioning*, and the terms are often used interchangeably. Consider the [Horizontal, vertical, and functional data partitioning guidance](#).

The Sharding pattern can scale to very large numbers of tenants. Additionally, depending on your workload, you might be able to achieve a high density of tenants to shards, so the cost can be attractive. The Sharding pattern can also be used to address [Azure subscription and service quotas, limits and constraints](#).

Some data stores, such as Azure Cosmos DB, provide native support for sharding or partitioning. When working with other solutions, such as Azure SQL, it can be more complex to build a sharding infrastructure and to route requests to the correct shard, for a given tenant.

Multitenant app with dedicated databases for each tenant

Another common approach is to deploy a single multitenant application, with dedicated databases for each tenant.



In this model, each tenant's data is isolated from the others, and you might be able to support some degree of customization for each tenant.

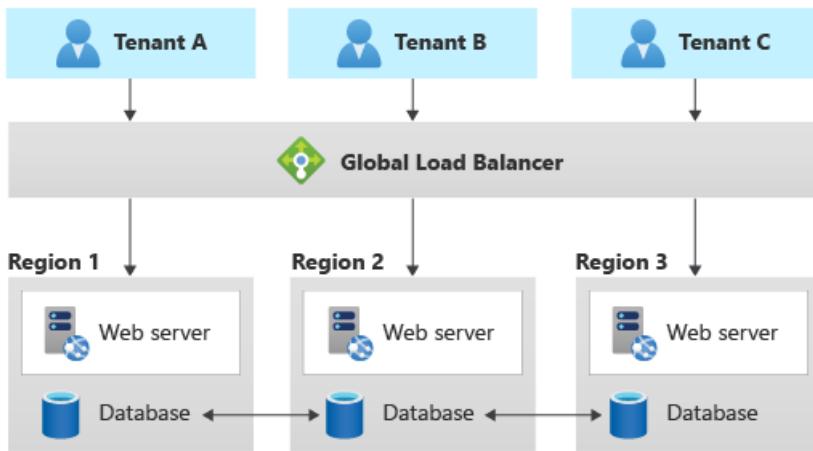
Because you provision dedicated data resources for each tenant, the cost for this approach can be higher than shared hosting models. However, Azure provides several options you can consider, in order to share the cost of hosting individual data resources across multiple tenants. For example, when you work with Azure SQL, you can consider [elastic pools](#). For Azure Cosmos DB, you can [provision throughput for a database](#) and the throughput is shared between the containers in that database, although this approach is not appropriate when you need guaranteed performance for each container.

In this approach, because only the data components are deployed individually for each tenant, you likely can achieve high density for the other components in your solution and reduce the cost of those components.

It's important to use automated deployment approaches when you provision databases for each tenant.

Geodes pattern

The [Geode pattern](#) is designed specifically for geographically distributed solutions, including multitenant solutions. It supports high load and high levels of resiliency. When working with the Geode pattern, the data tier must be able to replicate the data across geographic regions, and it should support multi-geography writes.



Azure Cosmos DB provides [multi-master writes](#) to support this pattern, and Cassandra supports multi-region clusters. Other data services are generally not able to support this pattern, without significant customization.

Antipatterns to avoid

When working with multitenant data services, it's important to avoid situations that inhibit your ability to scale.

For relational databases, these include:

- **Table-based isolation.** When you work within a single database, avoid creating individual tables for each tenant. A single database won't be able to support very large numbers of tenants when you use this approach, and it becomes increasingly difficult to query, manage, and update data. Instead, consider using a single set of multitenant tables with a tenant identifier column. Alternatively, you can use one of the patterns described above to deploy separate databases for each tenant.
- **Column-level tenant customization.** Avoid applying schema updates that only apply to a single tenant. For example, suppose you have a single multitenant database. Avoid adding a new column to meet a specific tenant's requirements. It might be acceptable for a small number of customizations, but this rapidly becomes unmanageable when you have a large number of customizations to consider. Instead, consider revising your data model to track custom data for each tenant in a dedicated table.
- **Manual schema changes.** Avoid updating your database schema manually, even if you only have a single shared database. It's easy to lose track of the updates you've applied, and if you need to scale out to more databases, it's challenging to identify the correct schema to apply. Instead, build an automated pipeline to deploy your schema changes, and use it consistently. Track the schema version used for each tenant in a dedicated database or lookup table.
- **Version dependencies.** Avoid having your application take a dependency on a single version of your database schema. As you scale, you may need to apply schema updates at different times for different tenants. Instead, ensure your application version is backwards-compatible with at least one schema version, and avoid destructive schema updates.

Databases

There are some features that can be useful for multitenancy. However, these aren't available in all database services. Consider whether you need these, when you decide on the service to use for your scenario:

- **Row-level security** can provide security isolation for specific tenants' data in a shared multitenant database. This feature is available in Azure SQL and Postgres Flex, but it's not available in other databases, like MySQL or Azure Cosmos DB.
- **Tenant-level encryption** might be required to support tenants that provide their own encryption keys for their data. This feature is available in Azure SQL as part of [Always Encrypted](#). Cosmos DB provides [customer-managed keys at the account level](#) and also [supports Always Encrypted](#).
- **Resource pooling** provides the ability to share resources and cost, between multiple databases or

containers. This feature is available in Azure SQL's [elastic pools](#) and [managed instances](#) and in Azure Cosmos DB's [database throughput](#), although each option has limitations you should be aware of.

- **Sharding and partitioning** has stronger native support in some services than others. This feature is available in Azure Cosmos DB, by using its [logical and physical partitioning](#), and in [Postgres Hyperscale](#). While Azure SQL doesn't natively support sharding, it provides [sharding tools](#) to support this type of architecture.

Additionally, when working with relational databases or other schema-based databases, consider where the schema upgrade process should be triggered, when you maintain a fleet of databases. In a small estate of databases, you might consider using a deployment pipeline to deploy schema changes. As you grow, it might be better for your application tier to detect the schema version for a specific database and to initiate the upgrade process.

File and blob storage

Consider the approach you use to isolate data within a storage account. For example, you might deploy separate storage accounts for each tenant, or you might share storage accounts and deploy individual containers.

Alternatively, you might create shared blob containers, and then you can use the blob path to separate data for each tenant. Consider [Azure subscription limits and quotas](#), and carefully plan your growth to ensure your Azure resources scale to support your future needs.

If you use shared containers, plan your authentication and authorization strategy carefully, to ensure that tenants can't access each other's data. Consider the [Valet Key pattern](#), when you provide clients with access to Azure Storage resources.

Cost allocation

Consider how you'll [measure consumption and allocate costs to tenants](#), for the use of shared data services. Whenever possible, aim to use built-in metrics instead of calculating your own. However, with shared infrastructure, it becomes hard to split telemetry for individual tenants. Application-level custom metering needs to be considered.

In general, cloud-native services, like Azure Cosmos DB and Azure Blob Storage, provide more granular metrics to track and model the usage for a specific tenant. For example, Azure Cosmos DB provides the consumed throughput for every request and response.

Next steps

For more information about multitenancy and specific Azure services, see:

- [Multitenancy and Azure Storage](#)
- [Multitenancy and Azure SQL Database](#)
- [Multitenancy and Azure Cosmos DB](#)

Architectural approaches for messaging in multitenant solutions

3/10/2022 • 24 minutes to read • [Edit Online](#)

Asynchronous messaging and event-driven communication are critical assets when building a distributed application that's composed of several internal and external services. When you design a multitenant solution, it's crucial to conduct a preliminary analysis to define how to share or partition messages that pertain to different tenants.

Sharing the same messaging system or event-streaming service can significantly reduce the operational cost and management complexity. However, using a dedicated messaging system for each tenant provides better data isolation, reduces the risk of data leakage, eliminates the [Noisy Neighbor issue](#), and allows to charge back Azure costs to tenants easily.

In this article, you can find a distinction between messages and events, and you'll find guidelines that solution architects can follow when deciding which approach to use for a messaging or eventing infrastructure in a multitenant solution.

Messages, data points, and discrete events

All messaging systems have similar functionalities, transport protocols, and usage scenarios. For example, most of the modern messaging systems support asynchronous communications that use volatile or persistent queues, AMQP and HTTPS transport protocols, at-least-once delivery, and so on. However, by looking more closely at the type of published information and how data is consumed and processed by client applications, we can distinguish between different kinds of messages and events. There's an essential distinction between services that deliver an event and systems that send a message. For more information, see the following resources:

- [Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus](#)
- [Events, Data Points, and Messages - Choosing the right Azure messaging service for your data](#)

Events

An event is a lightweight notification of a condition or a state change. Events can be discrete units or part of a series. Events are messages that don't generally convey a publisher's intent other than to inform. An event captures a fact and communicates it to other services or components. A consumer of the event can process the event as it pleases and doesn't fulfill any specific expectations the publisher holds. We can classify events into two main categories:

- **Discrete events** hold information about specific actions that the publishing application has carried out. When using asynchronous event-driven communication, an application publishes a notification event when something happens within its domain. One or more consuming applications needs to be aware of this state change, like a price change in a product catalog application. Consumers subscribe to the events to receive them asynchronously. When a given event happens, the consuming applications might update their domain entities, which can cause more integration events to be published.
- **Series events** carry informational data points, such as temperature readings from devices for analysis or user actions in a click-analytics solution, as elements in an ongoing, continuous stream. An event stream is related to a specific context, like the temperature or humidity reported by a field device. All the events related to the same context have a strict temporal order that matters when processing these events with an event stream processing engine. Analyzing streams of events that carry data points requires accumulating these events in a buffer that spans a desired time window. Or it can be in a selected number of items and then

processing these events, by using a near-real-time solution or machine-trained algorithm. The analysis of a series of events can yield signals, like the average of a value measured over a time window that crosses a threshold. Those signals can then be raised as discrete, actionable events.

Discrete events report state changes and are actionable. The event payload has information about what happened, but, in general, it doesn't have the complete data that triggered the event. For example, an event notifies consumers that a reporting application created a new file in a storage account. The event payload might have general information about the file, but it doesn't have the file itself. Discrete events are ideal for serverless solutions that need to scale.

Series events report a condition and are analyzable. Discrete events are time-ordered and interrelated. In some contexts, the consumer needs to receive the complete sequence of events to analyze what happened and to take the necessary action.

Messages

Messages contain raw data that's produced by a service to be consumed or stored elsewhere. Messages often carry information necessary for a receiving service to execute steps in a workflow or a processing chain. An example of this kind of communication is the [Command pattern](#). The publisher may also expect the receiver(s) of a message to execute a series of actions and to report back the outcome of their processing with an asynchronous message. A contract exists between the message publisher and message receiver(s). For example, the publisher sends a message with some raw data and expects the consumer to create a file from that data and send back a response message when done. In other situations, a sender process could send an asynchronous, one-way message to ask another service to perform a given operation, but with no expectation of getting back an acknowledgment or response message from it.

This kind of contractual message handling is quite different from a publisher who's publishing discrete events to an audience of consumer agents, without any specific expectation of how they will handle these notifications.

Example scenarios

Here is a list of some example multitenant scenarios for messages, data points, and discrete events:

- Discrete events:
 - A music-sharing platform tracks the fact that a specific user in a specific tenant has listened to a particular music track.
 - In an online store web application, the catalog application sends an event using the [Publisher-Subscriber pattern](#) to other applications to notify them that an item price has changed.
 - A manufacturing company pushes real-time information to customers and third parties about equipment maintenance, systems health, and contract updates.
- Data points:
 - A building control system receives telemetry events, such as temperature or humidity readings from sensors that belong to multiple customers.
 - An event-driven monitoring system receives and processes diagnostics logs in a near-real-time fashion from multiple services, such as web servers.
 - A client-side script on a web page collects user actions and sends them to a click-analytics solution.
- Messages:
 - A B2B finance application receives a message to begin processing a tenant's banking records.
 - A long-running workflow receives a message that triggers the execution of the next step.
 - The basket application of an online store application sends a CreateOrder command, by using an asynchronous, persistent message to the ordering application.

Key considerations and requirements

The [deployment and tenancy model](#) that you choose for your solution has a deep impact on security,

performance, data isolation, management, and the ability to cross-charge resource costs to tenants. This analysis includes the model that you select for your messaging and eventing infrastructure. In this section, we review some of the key decisions you must make when you plan for a messaging system in your multitenant solution.

Scale

The number of tenants, the complexity of message flows and event streams, the volume of messages, the expected traffic profile, and the isolation level should drive the decisions when planning for a messaging or eventing infrastructure.

The first step consists in conducting exhaustive capacity planning and establishing the necessary maximum capacity for a messaging system in terms of throughput to properly handle the expected volume of messages under regular or peak traffic.

Some service offerings, such as the [Azure Service Bus Premium tier](#), provide resource isolation at the CPU and memory level so that each customer workload runs in isolation. This resource container is called a *messaging unit*. Each premium namespace is allocated at least one messaging unit. You can purchase 1, 2, 4, 8, or 16 messaging units for each Service Bus Premium namespace. A single workload or entity can span multiple messaging units, and you can change the number of messaging units as necessary. The result is a predictable and repeatable performance for your Service Bus-based solution. For more information, see [Service Bus Premium and Standard messaging tiers](#). Likewise, Azure Event Hubs pricing tiers allow you to size the namespace, based on the expected volume of inbound and outbound events. For example, the Premium offering is billed by [processing units](#) (PUs), which correspond to a share of isolated resources (CPU, memory, and storage) in the underlying infrastructure. The effective ingest and stream throughput per PU will depend on the following factors:

- Number of producers and consumers
- Payload size
- Partition count
- Egress request rate
- Usage of Event Hubs Capture, Schema Registry, and other advanced features

For more information, see [Overview of Event Hubs Premium](#).

When your solution handles a considerable number of tenants, and you decide to adopt a separate messaging system for each tenant, you need to have a consistent strategy to automate the deployment, monitoring, alerting, and scaling of each infrastructure, separately from one other.

For example, a messaging system for a given tenant could be deployed during the provisioning process using an infrastructure as code (IaC) tool such as Terraform, Bicep, or Azure Resource Manager (ARM) JSON templates and a DevOps system such as Azure DevOps or GitHub Actions. For more information, see [Architectural approaches for the deployment and configuration of multitenant solutions](#).

The messaging system could be sized with a maximum throughput in messages per unit of time. If the system supports dynamic autoscaling, its capacity could be increased or decreased automatically, based on the traffic conditions and metrics to meet the expected service-level agreement.

Performance predictability and reliability

When designing and building a messaging system for a limited number of tenants, using a single messaging system could be an excellent solution to meet the functional requirements, in terms of throughput, and it could reduce the total cost of ownership. A multitenant application might share the same messaging entities, such as queues and topics across multiple customers. Or they might use a dedicated set of components for each, in order to increase tenant isolation. On the other hand, sharing the same messaging infrastructure across multiple tenants could expose the entire solution to the [Noisy Neighbor issue](#). The activity of one tenant could harm other tenants, in terms of performance and operability.

In this case, the messaging system should be properly sized to sustain the expected traffic load at peak time. Ideally, it should support autoscaling. The messaging system should dynamically scale out when the traffic increases and scale in when the traffic decreases. A dedicated messaging system for each tenant could also mitigate the Noisy Neighbor risk, but managing a large number of messaging systems could increase the complexity of the solution.

A multitenant application can eventually adopt a hybrid approach, where core services use the same set of queues and topics in a single, shared messaging system, in order to implement internal, asynchronous communications. In contrast, other services could adopt a dedicated group of messaging entities or even a dedicated messaging system, for each tenant to mitigate the Noisy Neighbor issue and guarantee data isolation.

When deploying a messaging system to virtual machines, you should co-locate these virtual machines with the compute resources, to reduce the network latency. These virtual machines should not be shared with other services or applications, to avoid the messaging infrastructure to compete for the system resources, such as CPU, memory, and network bandwidth with other systems. Virtual machines can also spread across the [availability zones](#), to increase the intra-region resiliency and reliability of business-critical workloads, including messaging systems. Suppose you decide to host a messaging system, such as [RabbitMQ](#) or [Apache ActiveMQ](#), on virtual machines. In that case, we recommend deploying it to a virtual machine scale set and configuring it for autoscaling, based on metrics such as CPU or memory. This way, the number of agent nodes hosting the messaging system will properly scale out and scale in, based on traffic conditions. For more information, see [Overview of autoscale with Azure virtual machine scale sets](#).

Likewise, when hosting your messaging system on a Kubernetes cluster, consider using node selectors and taints to schedule the execution of its pods on a dedicated node pool, not shared with other workloads, in order to avoid the [Noisy Neighbor issue](#). When deploying a messaging system to a zone-redundant AKS cluster, use [Pod topology spread constraints](#) to control how pods are distributed across your AKS cluster among failure-domains, such as regions, availability zones, and nodes. When hosting a third-party messaging system on AKS, use Kubernetes autoscaling to dynamically scale out the number of worker nodes when the traffic increases. With [Horizontal Pod Autoscaler](#) and [AKS cluster autoscaler](#), node and pod volumes get adjusted dynamically in real-time, to match the traffic condition and to avoid downtimes that are caused by capacity issues. For more information, see [Automatically scale a cluster to meet application demands on Azure Kubernetes Service \(AKS\)](#). Consider using [Kubernetes Event-Driven Autoscaling \(KEDA\)](#), if you want to scale out the number of pods of a Kubernetes-hosted messaging system, such as [RabbitMQ](#) or [Apache ActiveMQ](#), based on the length of a given queue. With KEDA, you can drive the scaling of any container in Kubernetes, based on the number of events needing to be processed. For example, you can use KEDA to scale applications based on the length of an [Azure Service Bus queue](#), of a [RabbitMQ queue](#), or an [ActiveMQ queue](#). For more information on KEDA, see [Kubernetes Event-driven Autoscaling](#).

Isolation

When designing the messaging system for a multitenant solution, you should consider that different types of applications require a different kind of isolation, which is based on the tenants' performance, privacy, and auditing requirements.

- **Multiple tenants can share the same messaging entities, such as queues, topics, and subscriptions, in the same messaging system.** For example, a multitenant application could receive messages that carry data for multiple tenants, from a single [Azure Service Bus](#) queue. This solution can lead to performance and scalability issues. If some of the tenants generate a larger volume of orders than others, this could cause a backlog of messages. This problem, also known as the [Noisy Neighbor issue](#), can degrade the service level agreement in terms of latency for some tenants. The problem is more evident if the consumer application reads and processes messages from the queue, one at a time, in a strict chronological order, and if the time necessary to process a message depends on the number of items in the payload. When sharing one or more queue resources across multiple tenants, the messaging infrastructure and processing systems should be able to accommodate the scale and throughput requirements-based expected traffic. This architectural approach is well-suited in those scenarios where a multitenant solution adopts a single pool of

worker processes, in order to receive, process, and send messages for all the tenants.

- **Multiple tenants share the same messaging system but use different topic or queue resources.** This approach provides better isolation and data protection at a higher cost, increased operational complexity, and lower agility because system administrators have to provision, monitor, and maintain a higher number of queue resources. This solution ensures a consistent and predictable experience for all tenants, in terms of performance and service-level agreements, as it prevents any tenant from creating a bottleneck that can impact the other tenants.
- **A separate, dedicated messaging system is used for each tenant.** For example, a multitenant solution can use a distinct [Azure Service Bus](#) namespace for each tenant. This solution provides the maximum degree of isolation, at a higher complexity and operational cost. This approach guarantees consistent performance and allows for fine-tuning the messaging system, based on specific tenant needs. When a new tenant is onboarded, in addition to a fully dedicated messaging system, the provisioning application might also need to create a separate managed identity or a service principal with the proper RBAC permissions to access the newly created namespace. For example, when a Kubernetes cluster is shared by multiple instances of the same SaaS application, one for each tenant, and each running in a separate namespace, each application instance may use a different service principal or managed identity to access a dedicated messaging system. In this context, the messaging system could be a fully-managed PaaS service, such as an [Azure Service Bus](#) namespace, or a Kubernetes-hosted messaging system, such as [RabbitMQ](#). When deleting a tenant from the system, the application should remove the messaging system and any dedicated resource that was priorly created for the tenant. The main disadvantage of this approach is that it increases operational complexity and reduces agility, especially when the number of tenants grows over time.

Review the following additional considerations and observations:

- If tenants need to use their own key to encrypt and decrypt messages, a multitenant solution should provide the option to adopt a separate Azure Service Bus Premium namespace for each tenant. For more information, see [Configure customer-managed keys for encrypting Azure Service Bus data at rest](#).
- If tenants need a high level of resiliency and business continuity, a multitenant solution should provide the ability to provision a Service Bus Premium namespace with geo-disaster recovery and [availability zones](#) enabled. For more information, see [Azure Service Bus Geo-disaster recovery](#).
- When using separate queue resources or a dedicated messaging system for each tenant, it's reasonable to adopt a separate pool of worker processes, for each of them to increase the data isolation level and reduce the complexity of dealing with multiple messaging entities. Each instance of the processing system could adopt different credentials, such as a connection string, a service principal, or a managed identity, in order to access the dedicated messaging system. This approach provides a better security level and isolation between tenants, but it requires an increased complexity in identity management.

Likewise, an event-driven application can provide different levels of isolation:

- An event-driven application receives events from multiple tenants, via a single, shared [Azure Event Hubs](#) instance. This solution provides a high level of agility, because onboarding a new tenant does not require creating a dedicated event-ingestion resource, but it provides a low data protection level, as it inter-mingles messages from multiple tenants in the same data structure.
- Tenants can opt for a dedicated event hub or Kafka topic to avoid the [Noisy Neighbor issue](#) and to meet their data isolation requirements, when events must not be co-mingled with those of other tenants, for security and data protection.
- If tenants need a high level of resiliency and business continuity, a multitenant should provide the ability to provision an Event Hubs Premium namespace, with geo-disaster recovery and [availability zones](#) enabled. For more information, see [Azure Event Hubs - Geo-disaster recovery](#).
- Dedicated Event Hubs, with Event Hubs Capture enabled, should be created for those customers that need to archive events to an Azure Storage Account, for regulatory compliance reasons and obligations. For more information, see [Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage](#).

- A single multitenant application can send notifications to multiple internal and external systems, by using [Azure Event Grid](#). In this case, a separate Event Grid subscription should be created for each consumer application. If your application makes use of multiple Event Grid instances to send notifications to a large number of external organizations, consider using *event domains*, which allow an application to publish events to thousands of topics, such as one for each tenant. For more information, see [Understand event domains for managing Event Grid topics](#).

Complexity of implementation

When designing a multitenant solution, it's essential to consider how the system will evolve in the medium to long term, in order to prevent its complexity from growing over time, until it is necessary to redesign part of or the entire solution. This redesign will help you cope with an increasing number of tenants. When designing a messaging system, you should consider the expected growth in message volumes and tenants, in the next few years, and then create a system that can scale out, in order to keep up with the predicted traffic and to reduce the complexity of operations, such as provisioning, monitoring, and maintenance.

The solution should automatically create or delete the necessary messaging entities any time that a tenant application is provisioned or unprovisioned, without the need for manual operations.

A particular concern for multitenant data solutions is the level of customization that you support. Any customization should be automatically configured and applied by the application provisioning system (such as a DevOps system), which is based on a set of initial parameters, whenever a single-tenant or multitenant application is deployed.

Complexity of management and operations

Plan from the beginning how you intend to operate, monitor, and maintain your messaging and eventing infrastructure and how your multitenancy approach affects your operations and processes. For example, consider the following possibilities:

- You might plan to host the messaging system that's used by your application in a dedicated set of virtual machines, one for each tenant. If so, how do you plan to deploy, upgrade, monitor, and scale out these systems?
- Likewise, if you containerize and host your messaging system in a shared Kubernetes cluster, how do you plan to deploy, upgrade, monitor, and scale out individual messaging systems?
- Suppose you share a messaging system across multiple tenants. How can your solution collect and report the per-tenant usage metrics or throttle the number of messages that each tenant can send or receive?
- When your messaging system leverages a PaaS service, such as Azure Service Bus, you should ask the following question:
 - How can you customize the pricing tier for each tenant, based on the features and isolation level (shared or dedicated) that are requested by the tenant?
 - How can you create a per-tenant managed identity and Azure role assignment to assign the proper permissions only to the messaging entities, such as queues, topics, and subscriptions, that the tenant can access? For more information, see [Authenticate a managed identity with Azure Active Directory to access Azure Service Bus resources](#).
- How will you migrate tenants, if they need to move to a different type of messaging service, a different deployment, or another region?

Cost

Generally, the higher the density of tenants to your deployment infrastructure, the lower the cost to provision that infrastructure. However, shared infrastructure increases the likelihood of issues like the [Noisy Neighbor issue](#), so consider the tradeoffs carefully.

Approaches and patterns to consider

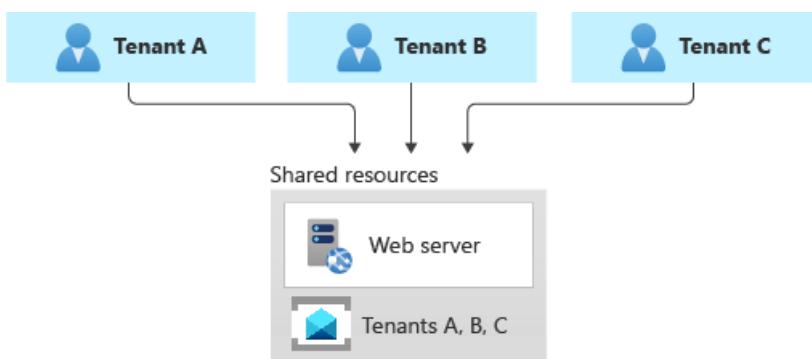
Several [Cloud Design Patterns](#) from the Azure Architecture Center can be applied to a multitenant messaging system. You might choose to follow one or more patterns consistently, or you could consider mixing and matching patterns, based on your needs. For example, you might use a multitenant Service Bus namespace for most of your tenants, but then you might deploy a dedicated, single-tenant Service Bus namespace for those tenants who pay more or who have higher requirements, in terms of isolation and performance. Similarly, it's often a good practice to scale by using deployment stamps, even when you use a multitenant Service Bus namespace or dedicated namespaces within a stamp.

Deployment Stamps pattern

For more information about the Deployment Stamps pattern and multitenancy, see [the Deployment Stamps pattern section of Architectural approaches for multitenancy](#). For more information about tenancy models, see [Tenancy models to consider for a multitenant solution](#).

Shared messaging system

You might consider deploying a shared messaging system, such as Azure Service Bus, and sharing it across all of your tenants.



This approach provides the highest density of tenants to the infrastructure, so it reduces the overall total cost of ownership. It also often reduces the management overhead, since there's a single messaging system or resource to manage and secure.

However, when you share a resource or an entire infrastructure across multiple tenants, consider the following caveats:

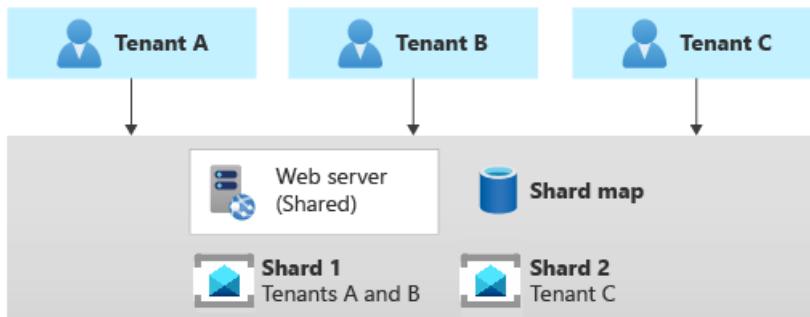
- Always keep in mind and consider the constraints, scaling capabilities, quotas, and limits of the resource in question. For example, the maximum number of [Service Bus namespaces in an Azure subscription](#), the maximum number of [Event Hubs in a single namespace](#), or the maximum throughput limits, might eventually become a hard blocker, if and when your architecture grows to support more tenants. Carefully consider the maximum scale that you need to achieve in terms of the number of namespaces per single Azure subscription, or queues per single namespace. Then compare your current and future estimates to the existing quotas and limits of the messaging system of choice, before you select this pattern.
- As mentioned in the above sections, the [Noisy Neighbor problem](#) might become a factor, when you share a resource across multiple tenants, especially if some are particularly busy or if they generate higher traffic than others. In this case, consider applying the [Throttling pattern](#) or the [Rate Limiting pattern](#) to mitigate these effects. For example, you could limit the maximum number of messages that a single tenant can send or receive in the unit of time.
- You might have difficulty monitoring the activity and [measuring the resource consumption](#) for a single tenant. Some services, such as Azure Service Bus, charge for messaging operations. Hence, when you share a namespace across multiple tenants, your application should be able to keep track of the number of messaging operations done on behalf of each tenant and the chargeback costs to them. Other services don't provide the same level of detail.

Tenants may have different requirements for security, intra-region resiliency, disaster recovery, or location. If these don't match your messaging system configuration, you might not be able to accommodate them just with

a single resource.

Sharding pattern

The [Sharding pattern](#) involves deploying multiple messaging systems, called *shards*, which contain one or more tenants' messaging entities, such as queues and topics. Unlike deployment stamps, shards don't imply that the entire infrastructure is duplicated. You might shard messaging systems without also duplicating or sharding other infrastructure in your solution.



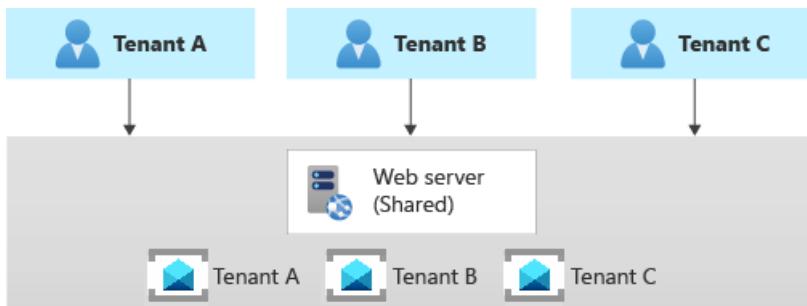
Every messaging system or *shard* can have different characteristics in terms of reliability, SKU, and location. For example, you could shard your tenants across multiple messaging systems with different characteristics, based on their location or needs in terms of performance, reliability, data protection, or business continuity.

When using the sharding pattern, you need to use a [sharding strategy](#), in order to map a given tenant to the messaging system that contains its queues. The [lookup strategy](#) uses a map to individuate the target messaging system, based on the tenant name or ID. Multiple tenants might share the same shard, but the messaging entities used by a single tenant won't be spread across multiple shards. The map can be implemented with a single dictionary that maps the tenant's name to the name or reference of the target messaging system. The map can be stored in a distributed cache accessible, by all the instances of a multitenant application, or in a persistent store, such as a table in a relational database or a table in a storage account.

The Sharding pattern can scale to very large numbers of tenants. Additionally, depending on your workload, you might be able to achieve a high density of tenants to shards, so the cost can be attractive. The Sharding pattern can also be used to address [Azure subscription and service quotas, limits, and constraints](#).

Multitenant app with dedicated messaging system for each tenant

Another common approach is to deploy a single multitenant application, with dedicated messaging systems for each tenant. In this tenancy model, you have some shared components, such as computing resources, while other services are provisioned, and managed using a single-tenant, dedicated deployment approach. For example, you could build a single application tier, and then deploy individual messaging systems for each tenant, as shown in the following illustration.



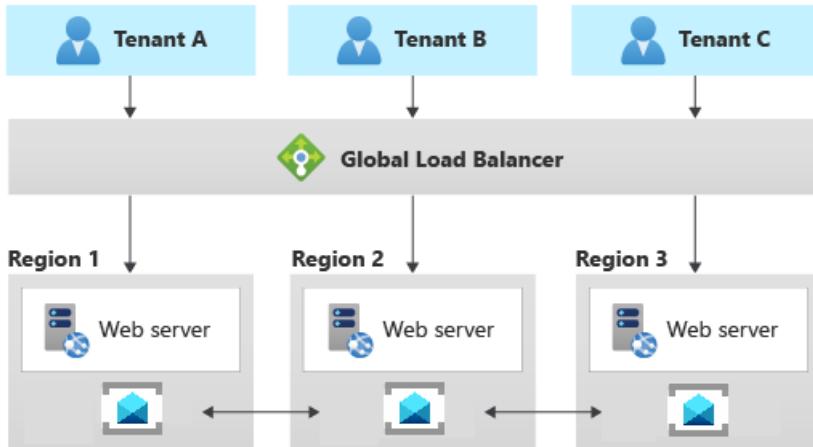
Horizontally partitioned deployments can help you mitigate a noisy-neighbor problem, if you've identified that most of the load on your system is due to specific components that you can deploy separately for each tenant. For example, you may need to use a separate messaging or event streaming system for each tenant because a single instance is not enough to keep up with traffic that's generated by multiple tenants. When using a dedicated messaging system for each tenant, if a single tenant causes a large volume of messages or events, this might affect the shared components but not other tenants' messaging systems.

Because you provision dedicated resources for each tenant, the cost for this approach can be higher than a shared hosting model. On the other hand, it's easier to charge back resource costs of a dedicated system to the unique tenant that makes use of it, when adopting this tenancy model. This approach allows you to achieve high density for other services, such as computing resources, and it reduces these components' costs.

With a horizontally partitioned deployment, you need to adopt an automated process for deploying and managing a multitenant application's services, especially those used by a single tenant.

Geodes pattern

The [Geode pattern](#) involves deploying a collection of backend services into a set of geographical nodes. Each can service any request for any client in any region. This pattern allows you to serve requests in an active-active style, which improves latency and increases availability, by distributing request processing around the globe.



[Azure Service Bus](#) and [Azure Event Hubs](#) support metadata disaster recovery, across primary and secondary disaster recovery namespaces and across separate regions and availability zones, in order to provide support for the best intra-region resiliency. Also, Azure Service Bus and Azure Event Hubs implement a set of federation patterns that actively replicate the same logical topic, queue, or event hub to be available in multiple namespaces, eventually located in different regions. For more information, see the following resources:

- [Message replication and cross-region federation](#)
- [Message replication tasks patterns](#)
- [Multi-site and multi-region federation](#)
- [Event replication tasks patterns](#)

Next steps

For more information about messaging design patterns, see the following resources:

- [Claim-Check pattern](#)
- [Competing Consumers pattern](#)
- [Event Sourcing pattern](#)
- [Pipes and Filters pattern](#)
- [Publisher-Subscriber pattern](#)
- [Sequential Convoy pattern](#)

Architectural approaches for AI and ML in multitenant solutions

3/10/2022 • 12 minutes to read • [Edit Online](#)

An ever-increasing number of multitenant solutions are built around artificial intelligence (AI) and machine learning (ML). A multitenant AI/ML solution is one that provides similar ML-based capabilities to any number of tenants. Tenants generally can't see or share the data of any other tenant, but in some situations, tenants might use the same models as other tenants.

Multitenant AI/ML architectures need to consider the requirements for data and models, as well as the compute resources that are required to train models and to perform inference from models. It's important to consider how multitenant AI/ML models are deployed, distributed, and orchestrated, and to ensure that your solution is accurate, reliable, and scalable.

Key considerations and requirements

When you work with AI and ML, it's important to separately consider your requirements for *training* and for *inference*. The purpose of training is to build a predictive model that's based on a set of data. You perform inference when you use the model to predict something in your application. Each of these processes has different requirements. In a multitenant solution, you should consider how your [tenancy model](#) affects each process. By considering each of these requirements, you can ensure that your solution provides accurate results, performs well under load, is cost-efficient, and can scale for your future growth.

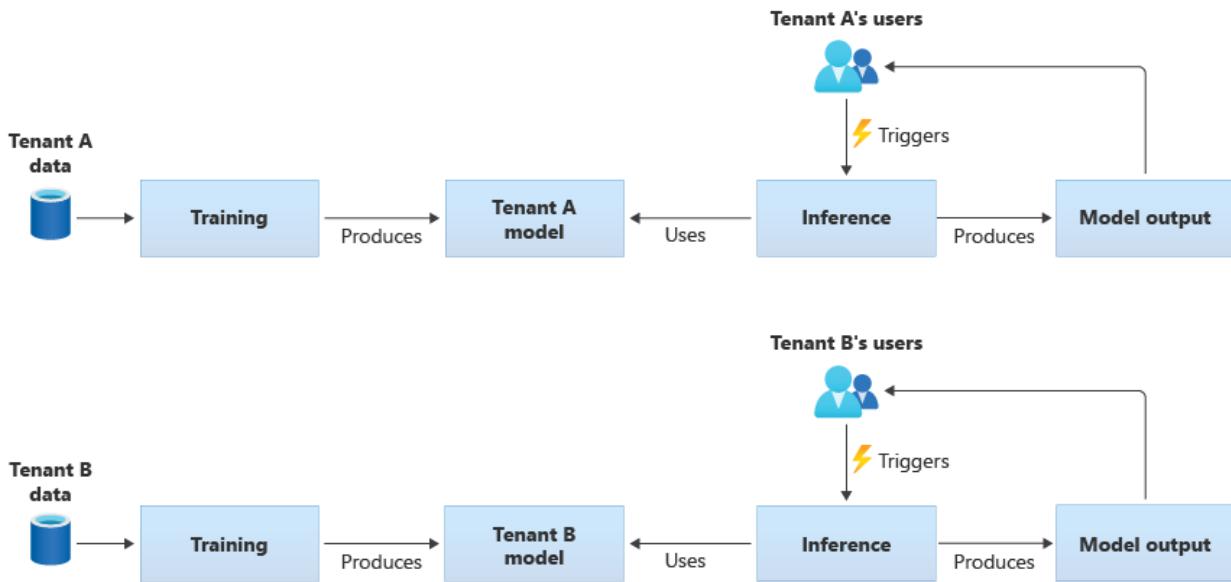
Tenant isolation

Ensure that tenants don't gain unauthorized or unwanted access to the data or models of other tenants. Treat models with a similar sensitivity to the raw data that trained them. Ensure that your tenants understand how their data is used to train models, and how models trained on other tenants' data might be used for inference purposes on their workloads.

There are three common approaches for working with ML models in multitenant solutions: tenant-specific models, shared models, and tuned shared models.

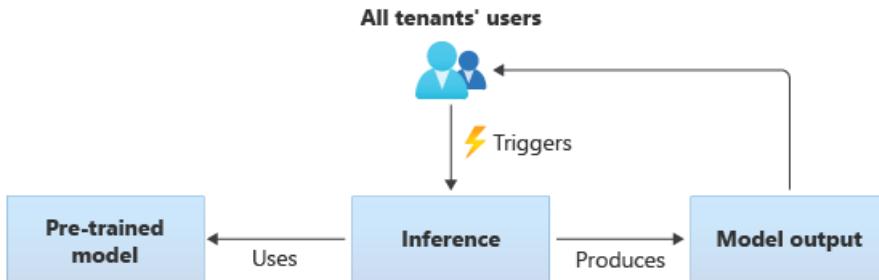
Tenant-specific models

Tenant-specific models are trained only on the data for a single tenant, and then they are applied to that single tenant. Tenant-specific models are appropriate when your tenants' data is sensitive, or when there's little scope to learn from the data that's provided by one tenant, and you apply the model to another tenant. The following diagram illustrates how you might build a solution with tenant-specific models for two tenants:

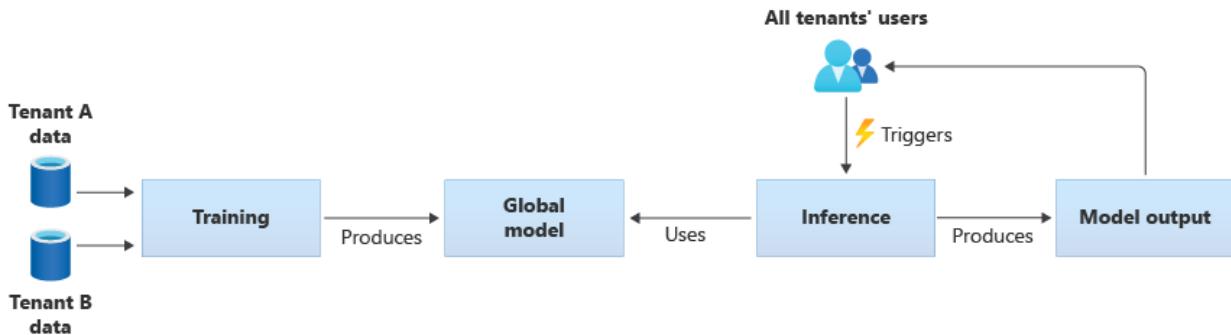


Shared models

In solutions that use shared models, all tenants perform inference based on the same shared model. Shared models might be pretrained models that you acquire or obtain from a community source. The following diagram illustrates how a single pretrained model can be used for inference by all tenants:



You also can build your own shared models by training them from the data provided by all of your tenants. The following diagram illustrates a single shared model, which is trained on data from all tenants:



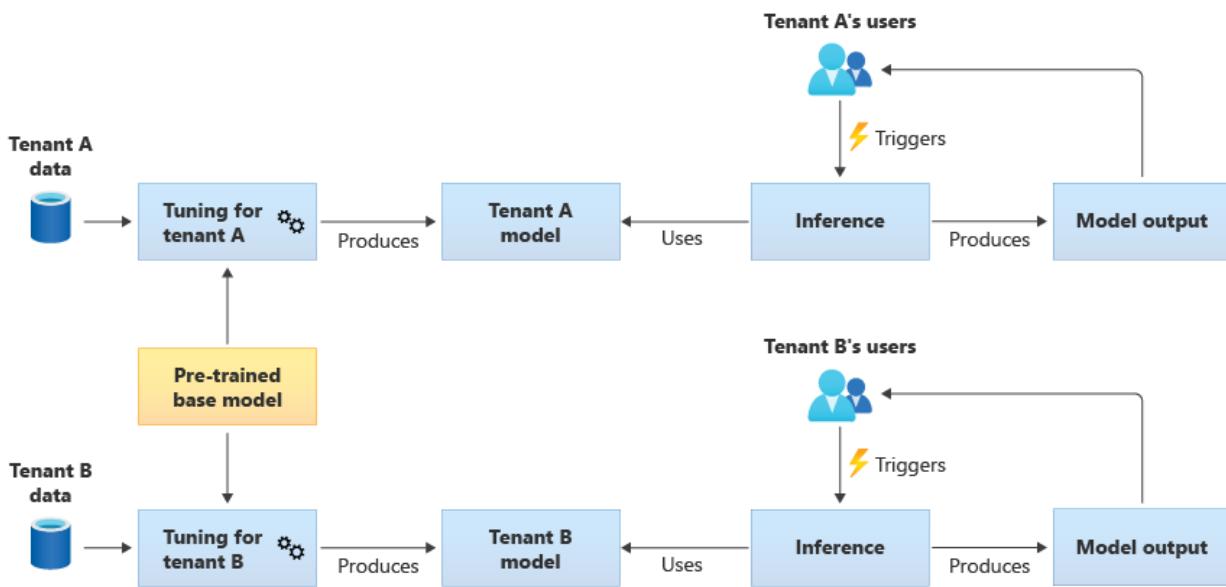
IMPORTANT

If you train a shared model from your tenants' data, ensure that your tenants understand and agree to the use of their data. Ensure identifying information is removed from your tenants' data.

Consider what to do, if a tenant objects to their data being used to train a model that will be applied to another tenant. For example, would you be able to exclude specific tenants' data from the training data set?

Tuned shared models

You also might choose to acquire a pretrained base model, and then perform further model tuning to make it applicable to each of your tenants, based on their own data. The following diagram illustrates this approach:



Scalability

Consider how the growth of your solution affects your use of AI and ML components. Growth can refer to an increase in the number of tenants, the amount of data stored for each tenant, the number of users, and the volume of requests to your solution.

Training: There are several factors that influence the resources that are required to train your models. These factors include the number of models you need to train, the amount of data that you train the models with, and the frequency at which you train or retrain models. If you create tenant-specific models, then as your number of tenants grows, the amount of compute resources and storage that you require will also be likely to grow. If you create shared models and train them based on data from all of your tenants, it's less likely that the resources for training will scale at the same rate as the growth in your number of tenants. However, an increase in the overall amount of training data will affect the resources that are consumed, to train both the shared and tenant-specific models.

Inference: The resources that are required for inference are usually proportional to the number of requests that access the models for inference. As the number of tenants increase, the number of requests is also likely to increase.

It's a good general practice to use Azure services that scale well. Because AI/ML workloads tend to make use of containers, Azure Kubernetes Service (AKS) and Azure Container Instances (ACI) tend to be common choices for AI/ML workloads. AKS is usually a good choice to enable high scale, and to dynamically scale your compute resources based on demand. For small workloads, ACI can be a simple compute platform to configure, although it doesn't scale as easily as AKS.

Performance

Consider the performance requirements for the AI/ML components of your solution, for both training and inference. It's important to clarify your latency and performance requirements for each process, so that you can measure and improve as required.

Training: Training is often performed as a batch process, which means that it might not be as performance-sensitive as other parts of your workload. However, you need to ensure that you provision sufficient resources to perform your model training efficiently, including as you scale.

Inference: Inference is a latency-sensitive process, often requiring a fast or even real-time response. Even if you don't need to perform inference in real time, ensure you monitor the performance of your solution and use the appropriate services to optimize your workload.

Consider using Azure's high-performance computing capabilities for your AI and ML workloads. Azure provides many different types of virtual machines and other hardware instances. Consider whether your solution would

benefit from using CPUs, [GPUs](#), [FPGAs](#), or other hardware-accelerated environments. Azure also provides real-time inference with NVIDIA GPUs, including NVIDIA Triton Inference Servers. For low-priority compute requirements, consider using [AKS spot node pools](#). To learn more about optimizing compute services in a multitenant solution, see [Architectural approaches for compute in multitenant solutions](#).

Model training typically requires a lot of interaction with your data stores, so it's also important to consider your data strategy and the performance that your data tier provides. For more information about multitenancy and data services, see [Architectural approaches for storage and data in multitenant solutions](#).

Consider profiling your solution's performance. For example, [Azure Machine Learning provides profiling capabilities](#) that you can use when developing and instrumenting your solution.

Implementation complexity

When you build a solution to use AI and ML, you can choose to use prebuilt components, or to build custom components. There are two key decisions you need to make. The first is the *platform or service* you use for AI and ML. The second is whether you use pretrained models or build your own custom models.

Platforms: There are many Azure services that you can use for your AI and ML workloads. For example, Azure Cognitive Services provides an API to perform inference against prebuilt models, and Microsoft manages the underlying resources. Azure Cognitive Services enables you to quickly deploy a new solution, but it gives you less control over how training and inference is performed, and it might not suit every type of workload. In contrast, Azure Machine Learning is a platform that enables you to build, train, and use your own ML models. Azure Machine Learning provides control and flexibility, but it increases the complexity of your design and implementation. Review the [machine learning products and technologies from Microsoft](#) to make an informed decision when selecting an approach.

Models: Even when you don't use a full model that's provided by a service like Azure Cognitive Services, you can still accelerate your development by using a pretrained model. If a pretrained model doesn't precisely suit your needs, consider extending a pretrained model by applying a technique called *transfer learning*. Transfer learning enables you to extend an existing model and apply it to a different domain. For example, if you're building a multitenant music recommendation service, you might consider building off a pretrained model of music recommendations, and use transfer learning to train the model for a specific user's music preferences.

By using a prebuilt ML platform like Azure Cognitive Services, or a pretrained model, you can significantly reduce your initial research and development costs. The use of prebuilt platforms might save you many months of research, and avoid the need to recruit highly qualified data scientists to train, design, and optimize models.

Cost optimization

Generally, AI and ML workloads incur the greatest proportion of their costs from the compute resources that are required for model training and inference. Review [Architectural approaches for compute in multitenant solutions](#) to understand how to optimize the cost of your compute workload for your requirements.

Consider the following requirements when planning your AI and ML costs:

- **Determine compute SKUs for training.** For example, refer to [guidance on how to do this with Azure ML](#).
- **Determine compute SKUs for inference.** For an example cost estimate for inference, [refer to the guidance for Azure ML](#).
- **Monitor your utilization.** By observing the utilization of your compute resources, you can determine whether you should decrease or increase their capacity by deploying different SKUs, or scale the compute resources as your requirements change. See [Azure Machine Learning Monitor](#).
- **Optimize your compute clustering environment.** When you use compute clusters, monitor cluster utilization or configure [autoscaling](#) to scale down compute nodes.
- **Share your compute resources.** Consider whether you can optimize the cost of your compute resources by sharing them across multiple tenants.
- **Consider your budget.** Understand whether you have a fixed budget, and monitor your consumption

accordingly. You can [set up budgets](#) to prevent overspending and to allocate quotas based on tenant priority.

Approaches and patterns to consider

Azure provides a set of services to enable AI and ML workloads. There are several common architectural approaches used in multitenant solutions: to use prebuilt AI/ML solutions, to build a custom AI/ML architecture by using Azure Machine Learning, and to use one of the Azure analytics platforms.

Use prebuilt AI/ML services

It's a good practice to try to use prebuilt AI/ML services, where you can. For example, your organization might be beginning to look at AI/ML and want to quickly integrate with a useful service. Or, you might have basic requirements that don't require custom ML model training and development. Prebuilt ML services enable you to use inference without building and training your own models.

Azure provides several services that provide AI and ML technology across a range of domains, including language and speech recognition, knowledge, document and form recognition, and computer vision. Azure's prebuilt AI/ML services include [Azure Cognitive Services](#), [Azure Cognitive Search](#), and [Azure Form Recognizer](#). Each service provides a simple interface for integration, and a collection of pretrained and tested models. As managed services, they provide service-level agreements and require little configuration or ongoing management. You don't need to develop or test your own models to use these services.

Many managed ML services don't require model training or data, so there's usually no tenant data isolation concerns. However, when you work with Cognitive Search in a multitenant solution, review [Design patterns for multitenant SaaS applications](#) and [Azure Cognitive Search](#).

Consider the scale requirements for the components in your solution. For example, many of the APIs within Azure Cognitive Services support a maximum number of requests per second. If you deploy a single Cognitive Services resource to share across your tenants, then as the number of tenants increases, you might need to [scale to multiple resources](#).

NOTE

Some managed services enable you to train with your own data, including the [Custom Vision service](#), the [Face API](#), and [Form Recognizer custom models](#). When you work with these services, it's important to consider the [isolation requirements](#) for your tenants' data.

Custom AI/ML architecture

If your solution requires custom models, or you work in a domain that isn't covered by a managed ML service, then consider building your own AI/ML architecture. [Azure Machine Learning](#) provides a suite of capabilities to orchestrate the training and deployment of ML models. Azure Machine Learning supports many open-source machine learning libraries, including [PyTorch](#), [Tensorflow](#), [Scikit](#), and [Keras](#). You can continuously monitor models' performance metrics, detect data drift, and trigger retraining to improve model performance.

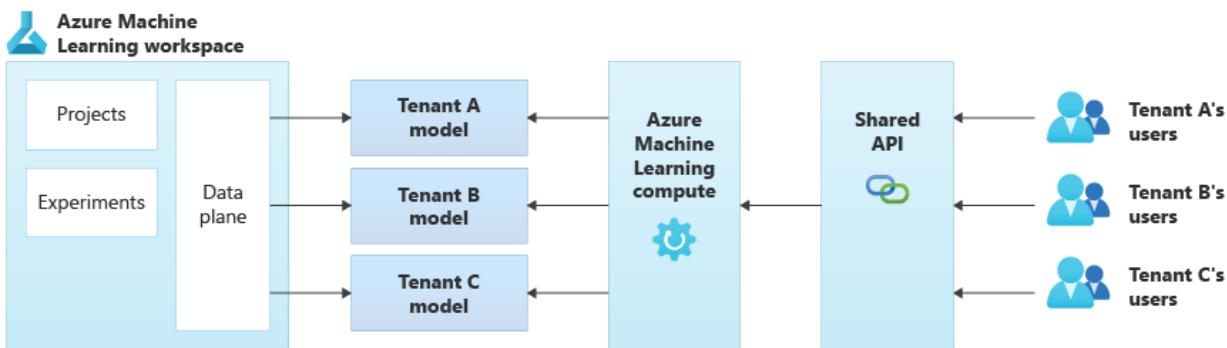
Throughout the lifecycle of your ML models, Azure Machine Learning enables auditability and governance with built-in tracking and lineage for all your ML artifacts.

When working in a multitenant solution, it's important to consider the [isolation requirements of your tenants](#) during both the training and inference stages. You also need to determine your model training and deployment process. Azure Machine Learning provides a pipeline to train models, and to deploy them to an environment to be used for inference. In a multitenant context, consider whether models should be deployed to shared compute resources, or if each tenant has dedicated resources. Design your model deployment pipelines, based on your [isolation model](#) and your [tenant deployment process](#).

When you use open-source models, you might need to retrain these models by using transfer learning or tuning. Consider how you will manage the different models and training data for each tenant, as well as versions

of the model.

The following diagram illustrates an example architecture that uses Azure Machine Learning. The example uses the [tenant-specific models](#) isolation approach.



Integrated AI/ML solutions

Azure provides several powerful analytics platforms that can be used for a range of purposes. These platforms include [Azure Synapse Analytics](#), [Databricks](#), and [Apache Spark](#).

You can consider using these platforms for AI/ML, when you need to scale your ML capabilities to a very large number of tenants, and when you need large-scale compute and orchestration. You also might consider using these platforms for AI/ML, when you need a broad analytics platform for other parts of your solution, such as for data analytics and integration with reporting through Microsoft Power BI. You can deploy a single platform that covers all of your analytics and AI/ML needs. When you implement data platforms in a multitenant solution, review [Architectural approaches for storage and data in multitenant solutions](#).

Antipatterns to avoid

- **Failure to consider isolation requirements.** It's important to carefully consider how you [isolate tenants' data and models](#), both for training and inference. Failing to do so might violate legal or contractual requirements. It also might reduce the accuracy of your models to train across multiple tenants' data, if the data is substantially different.
- **Noisy Neighbors.** Consider whether your training or inference processes could be subject to the [Noisy Neighbor problem](#). For example, if you have several large tenants and a single small tenant, ensure that the model training for the large tenants doesn't inadvertently consume all of the compute resources and starve the smaller tenants. Use resource governance and monitoring to mitigate the risk of a tenant's compute workload that's affected by the activity of the other tenants.

Next steps

Review [Architectural approaches for compute in multitenant solutions](#) approaches.

Service-specific guidance for a multitenant solution

3/10/2022 • 2 minutes to read • [Edit Online](#)

When you're building a solution on Azure, you combine multiple distinct Azure services together to achieve your business goals. Although Azure services work in a consistent manner, there are specific considerations for how you design and implement each service. When you design a multitenant solution, there are further considerations to review, for each service.

In this section, we provide guidance about the features of each service that are helpful for multitenant solutions. We also discuss the levels of tenant isolation that each service supports. Where applicable, we link to more details and sample implementations in the service's documentation.

NOTE

The content in this section focuses specifically on the aspects of each service that are useful when building a multitenant solution on Azure. For comprehensive information about each service and its features, refer to the service's documentation.

Intended audience

The content in this section is designed for architects, lead developers, and anyone building or implementing Azure components for a multitenant solution. The audience also includes independent software vendors (ISVs) and startups who develop SaaS solutions.

What's covered in this section?

The articles in this section describe some Azure services commonly used in multitenant solutions. The following Azure services are covered so far:

- Deployment and configuration: [Azure Resource Manager](#)
- Compute: [App Service](#) and [Azure Functions](#)
- Networking: [NAT Gateway](#)
- Storage and data: [Azure Storage](#), [Azure SQL Database](#), [Azure Cosmos DB](#), and [Azure Database for PostgreSQL](#)

We frequently add new articles with guidance for additional services. You're also welcome to [submit suggestions for additional service-specific guidance](#).

Next steps

Review the guidance for [Azure Resource Manager](#).

Multitenancy and Azure Resource Manager

3/10/2022 • 4 minutes to read • [Edit Online](#)

Azure Resource Manager is the core resource management service for Azure. Every resource in Azure is created, managed, and eventually deleted through Resource Manager. When you build a multitenant solution, you often work with Resource Manager to dynamically provision resources for each tenant. On this page, we describe some of the features of Resource Manager that are relevant to multitenant solutions. We'll also provide links to guidance that can help you when you're planning to use Resource Manager.

Features of Resource Manager that support multitenancy

Infrastructure as code

Resource Manager provides tooling to support infrastructure as code, sometimes referred to as IaC.

Infrastructure as code is important for all solutions in the cloud, but when working with multitenant solutions, it becomes particularly important. A multitenant solution often requires you to scale deployments and quickly provision new resources, as you onboard new tenants. If you manually create or configure resources, then you introduce extra risk and time to the process. It results in a less reliable deployment process overall.

When deploying your infrastructure as code from a deployment pipeline, we recommend you use [Bicep](#), which is a language specifically designed to deploy and manage Azure resources in a declarative way. You can also use [JSON Azure Resource Manager templates](#) (ARM templates), Terraform, or other third-party products that access the underlying Resource Manager APIs.

[Template specs](#) can be useful for provisioning new resources, deployment stamps, or environments from a single and well-parameterized template. By using template specs, you can create a central repository of the templates that you use to deploy your tenant-specific infrastructure. The templates are stored and managed within Azure itself, and you can reuse the template specs whenever you need to deploy from them.

In some solutions, you might choose to write custom code to dynamically provision or configure resources, or to initiate a template deployment. The [Azure SDKs](#) can be used from your own code, to manage your Azure environment. Ensure that you follow good practices around managing the authentication of your application to Resource Manager, and use [managed identities](#) wherever possible, to avoid storing and managing credentials.

Role-based access control

[Role-based access control](#) (Azure RBAC) provides you with a fine-grained approach to manage access to your Azure resources. In a multitenant solution, consider whether you have resources that should have specific Azure RBAC policies applied. For example, you might have some tenants with particularly sensitive data, and you might need to apply RBAC to grant access to those individuals, without including other people in your organization. Similarly, tenants might ask to access their Azure resources directly, such as during an audit. Should you choose to allow this, finely scoped RBAC permissions can enable you to grant access to a tenant's data, without providing access to another tenants' data.

Tags

[Tags](#) enable you to add custom metadata to your Azure resources, resource groups, and subscriptions. Consider tagging your tenant-specific resources with the tenant's identifier so that you can easily [track and allocate your Azure costs](#), and to simplify your resource management.

Azure resource quotas

Resource Manager is one of the points in Azure that enforces [limits and quotas](#). These quotas are important to consider throughout your design process. All Azure resources have limits that need to be adhered to, and these

limits include the number of requests that can be made against Resource Manager, within a certain time period. If you exceed this limit, [Resource Manager throttles the requests](#).

When you build a multitenant solution that performs automated deployments, you might reach these limits faster than other customers. Similarly, multitenant solutions that provision large amounts of infrastructure can trigger the limits.

Every Azure service is managed by a *resource provider*, which may also define its own limits. For example, the Azure Compute Resource Provider manages the provisioning of virtual machines, and [it defines limits on the number of requests](#) that can be made in a short period. Some other resource provider limits are documented in [Resource provider limits](#).

If you are at risk of exceeding the limits defined by Resource Manager or a resource provider, you can consider mitigations, such as the following:

- Shard your workload across subscriptions.
- Use multiple resource groups.
- Send requests from different Azure Active Directory (Azure AD) principals.
- Request additional quota allocations. In general, quota allocation requests are [submitted by opening a support case](#), although some services provide APIs for these requests, such as for [virtual machine reserved instances](#).

The mitigations you select need to be appropriate for the specific limit you encounter.

Isolation models

In some multitenant solutions, you might decide to deploy separate or dedicated resources for each tenant. Resource Manager provides several models that you can use to isolate resources, depending on your requirements and the reason you choose to isolate the resources. See [Azure resource organization in multitenant solutions](#) for guidance about how to isolate your Azure resources.

Next steps

Review [deployment and configuration approaches for multitenancy](#).

Multitenancy for Azure App Service and Azure Functions

3/10/2022 • 8 minutes to read • [Edit Online](#)

Azure App Service is a powerful web application hosting platform. Azure Functions, built on top of the App Service infrastructure, enables you to easily build serverless and event-driven compute workloads. Both services are frequently used in multitenant solutions.

Features of Azure App Service and Azure Functions that support multitenancy

Azure App Service and Azure Functions include many features that support multitenancy.

Custom domain names

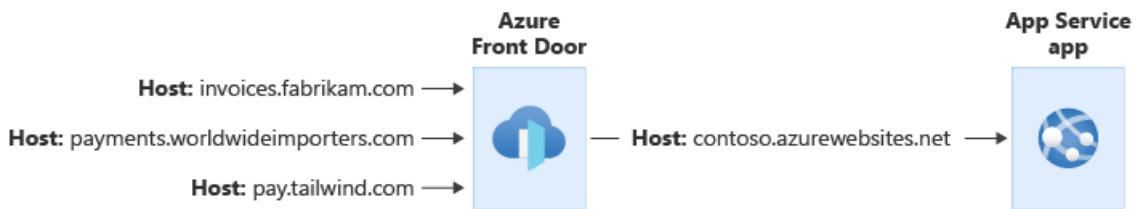
Azure App Service enables you to use [wildcard DNS](#) and to add your own [wildcard TLS certificates](#). When you use [tenant-specific subdomains](#), wildcard DNS and TLS certificates enable you to easily scale your solution to a large number of tenants, without requiring a manual reconfiguration for each new tenant.

When you use [tenant-specific custom domain names](#), you might have a large number of custom domain names that need to be added to your app. It can become cumbersome to manage a lot of custom domain names, especially when they require individual TLS certificates. App Service provides [managed TLS certificates](#), which reduces the work required when you work with custom domains. However, there are [limits to consider](#), such as how many custom domains can be applied to a single app.

Integration with Azure Front Door

App Service and Azure Functions integrate with [Azure Front Door](#), to act as the internet-facing component of your solution.

Azure Front Door enables you to add a web application firewall (WAF) and edge caching, and it provides other performance optimizations. You can easily reconfigure your traffic flows to direct traffic to different backends, based on changing business or technical requirements. When you use Azure Front Door with a multitenant app, you can use it to manage your custom domain names and to terminate your TLS connections. Your App Service application is then configured with a single hostname, and all traffic flows through to that application, which helps you avoid managing custom domain names in multiple places.



As in the above example, [Azure Front Door can be configured to modify the request's Host header](#). The original Host header sent by the client is propagated through the X-Forwarded-Host header, and your application code can use this header to [map the request to the correct tenant](#).

TIP

If your application sends cookies or redirection responses, you need to take special care. Changes in the request's [Host](#) headers might invalidate these responses.

You can use [private endpoints](#) or App Service [access restrictions](#) to ensure that traffic has flowed through Front Door before reaching your app.

Authentication and authorization

Azure App Service can [validate authentication tokens on behalf of your app](#). If a request doesn't contain a token, the token is invalid, or the request isn't authorized. App Service can be configured to block the request or to redirect to your identity provider, so that the user can sign in.

If your tenants use Azure Active Directory (Azure AD) as their identity system, you can configure Azure App Service to use [the /common endpoint](#) to validate user tokens. This ensures that, regardless of the user's Azure AD tenant, their tokens are validated and accepted.

You can also integrate Azure App Service with Azure AD B2C for authentication of consumers.

More information:

- [App Service authorization](#)
- [Configure authentication in a sample web app by using Azure AD B2C](#)
- [Working with multitenant Azure AD identities](#)

Access restrictions

You can restrict the traffic to your app by using [access restrictions](#). These can be used to specify the IP address ranges or the virtual networks that are allowed to connect to the app.

When you work with a multitenant solution, be aware of the maximum number of access restriction rules. For example, if you need to create an access restriction rule for every tenant, you might exceed the maximum number of rules that are allowed. If you need a larger number of rules, consider deploying a reverse proxy like [Azure Front Door](#).

Isolation models

When working with a multitenant system using Azure App Service or Azure Functions, you need to make a decision about the level of isolation that you want to use. Refer to the [tenancy models to consider for a multitenant solution](#) and to the guidance provided in the [architectural approaches for compute in multitenant solutions](#), to help you select the best isolation model for your scenario.

When you work with Azure App Service and Azure Functions, you should be aware of the following key concepts:

- In Azure App Service, a [plan](#) represents your hosting infrastructure. An app represents a single application running on that infrastructure. You can deploy multiple apps to a single plan.
- In Azure Functions, your hosting and application are also separated, but you have [additional hosting options available](#) for *elastic hosting*, where Azure Functions manages scaling for you. For simplicity, we refer to the hosting infrastructure as a [plan](#) throughout this article, because the principles described here apply to both App Service and Azure Functions, regardless of the hosting model you use.

Plans per tenant

The strongest level of isolation is to deploy a dedicated plan for a tenant. This dedicated plan ensures that the tenant has full use of all of the server resources that are allocated to that plan.

This approach enables you to scale your solution to provide performance isolation for each tenant, and to avoid the [Noisy Neighbor problem](#). However, it also has a higher cost because the resources aren't shared with multiple tenants. Also, you need to consider the [maximum number of plans](#) that can be deployed into a single Azure resource group.

Apps per tenant with shared plans

You can also choose to share your plan between multiple tenants, but deploy separate apps for each tenant. This provides you with logical isolation between each tenant, and this approach gives you the following advantages:

- **Cost efficiency:** By sharing your hosting infrastructure, you can generally reduce your overall costs per tenant.
- **Separation of configuration:** Each tenant's app can have its own domain name, TLS certificate, access restrictions, and token authorization policies applied.
- **Separation of upgrades:** Each tenant's application binaries can be upgraded independently of other apps on the same plan.

However, because the plan's compute resources are shared, the apps might be subject to the [Noisy Neighbor problem](#). Additionally, there are [limits to how many apps can be deployed to a single plan](#).

Shared apps

You can also consider deploying a shared application on a single plan. This tends to be the most cost-efficient option, and it requires the least operational overhead because there are fewer resources to manage. You can scale the overall plan based on load or demand, and all tenants sharing the plan will benefit from the increased capacity.

It's important to be aware of the [App Service quotas and limits](#), such as the maximum number of custom domains that can be added to a single app, and the maximum number of instances of a plan that can be provisioned.

To be able to use this model, your application code must be multitenancy-aware.

NOTE

Don't use [deployment slots](#) for different tenants. Slots don't provide resource isolation. They are designed for deployment scenarios when you need to have multiple versions of your app running for a short time, such as blue-green deployments and a canary rollout strategy.

Host APIs

You can host APIs on both Azure App Service and Azure Functions. Your choice of platform will depend on the specific feature set and scaling options you need.

Whichever platform you use to host your API, consider using [Azure API Management](#) in front of your API application. API Management provides many features that can be helpful for multitenant solutions, including the following:

- A centralized point for all [authentication](#). This might include determining the tenant identifier from a token claim or other request metadata.
- [Routing requests to different API backends](#), which might be based on the request's tenant identifier. This can be helpful when you host multiple [deployment stamps](#), with their own independent API applications, but you need to have a single API URL for all requests.

Networking and multitenancy

IP addresses

Many multitenant applications need to connect to tenants' on-premises networks to send data.

If you need to send outbound traffic from a known static IP address or from a set of known static IP addresses, consider using a [NAT Gateway](#). App Service provides [guidance on how to integrate with a NAT Gateway](#).

If you don't need a static outbound IP address, but instead you need to occasionally check the IP address that your application uses for outbound traffic, you can [query the current IP addresses of the App Service deployment](#).

Quotas

Because App Service is itself a multitenant service, you need to take care about how you use shared resources. Networking is an area that you need to pay particular attention to, because there are [limits](#) that affect how your application can work with both inbound and outbound network connections, including source network address translation (SNAT) and TCP port limits.

If your application connects to a large number of databases or external services, then your app might be at risk of [SNAT port exhaustion](#). In general, SNAT port exhaustion indicates that your code isn't correctly reusing TCP connections, and even in a multitenant solution, you should ensure you follow the recommended practices for reusing connections.

However, in some multitenant solutions, the number of outbound connections to distinct IP addresses can result in SNAT port exhaustion, even when you follow good coding practices. In these scenarios, [consider deploying NAT Gateway](#) to increase the number of SNAT ports that are available for your application to use, or use [service endpoints](#) when you connect to Azure services, to bypass load balancer limits. Even with these controls in place, you might approach limits with a large number of tenants, so you should plan to scale to additional App Service plans or [deployment stamps](#).

Next steps

Review [Resources for architects and developers of multitenant solutions](#).

Azure NAT Gateway considerations for multitenancy

3/10/2022 • 3 minutes to read • [Edit Online](#)

Azure NAT Gateway provides control over outbound network connectivity from your resources that are hosted within an Azure virtual network. In this article, we review how NAT Gateway can mitigate Source Network Address Translation (SNAT) port exhaustion, which can affect multitenant applications. We also review how NAT Gateway assigns static IP addresses to the outbound traffic from your multitenant solution.

NOTE

Firewalls, like [Azure Firewall](#), enable you to control and log your outbound traffic. Azure Firewall also provides similar SNAT port scale and outbound IP address control to NAT Gateway. NAT Gateway is less costly, but it also has fewer features and is not a security product.

Features of NAT Gateway that support multitenancy

High-scale SNAT ports

SNAT ports are allocated when your application makes multiple concurrent outbound connections to the same public IP address, on the same port. SNAT ports are a finite resource within [load balancers](#). If your application opens large numbers of separate connections to the same host, it can consume all of the available SNAT ports. This situation is called *SNAT port exhaustion*.

In most applications, SNAT port exhaustion indicates that your application is incorrectly handling HTTP connections or TCP ports. However, some multitenant applications are at particular risk of exceeding SNAT port limits, even if they reuse connections appropriately. For example, this situation can occur when your application connects to many tenant-specific databases behind the same database gateway.

TIP

If you observe SNAT port exhaustion in a multitenant application, you should verify whether [your application follows good practices](#). Ensure you reuse HTTP connections and don't recreate new connections every time you connect to an external service. You might be able to deploy a NAT Gateway to work around the problem, but if your code doesn't follow the best practices, you could encounter the problem again in the future.

The issue is exacerbated when you work with Azure services that share SNAT port allocations between multiple customers, such as [Azure App Service](#) and [Azure Functions](#).

If you determine you're experiencing SNAT exhaustion and are sure your application code correctly handles your outbound connections, consider deploying NAT Gateway. This approach is commonly used by customers who deploy multitenant solutions that are built on [Azure App Service](#) and [Azure Functions](#).

Each NAT gateway provides at least 64,000, and up to 1 million, SNAT ports. If you need to scale beyond this limit, you can consider [deploying multiple NAT Gateway instances across multiple subnets or VNets](#). Each virtual machine in a subnet can use any of the available SNAT ports, if it needs them.

Outbound IP address control

Outbound IP address control can be useful in multitenant applications, when you have all of the following requirements:

- You use Azure services that don't automatically provide dedicated static IP addresses for outbound traffic.

These services include Azure App Service, Azure Functions, API Management (when running in the consumption tier), and Azure Container Instances.

- You need to connect to your tenants' networks over the internet.
- Your tenants need to filter incoming traffic that's based on its IP address.

When a NAT Gateway instance is applied to a subnet, any outbound traffic from that subnet uses the public IP addresses that's associated with the NAT gateway.

NOTE

When you associate multiple public IP addresses with a single NAT Gateway, your outbound traffic could come from any of those IP addresses. You might need to configure firewall rules at the destination. You should either allow each IP address, or use a [public IP address prefix](#) resource to use a set of public IP addresses in the same range.

Isolation models

If you need to provide different outbound public IP addresses for each tenant, you must deploy individual NAT Gateway resources. Each subnet can be associated with a single NAT Gateway instance. To deploy more NAT gateways, you need to deploy multiple subnets or virtual networks. In turn, you likely need to deploy multiple sets of compute resources.

Review [Architectural approaches for networking in multitenant solutions](#) for more information about how to design a multitenant network topology.

Next steps

- [Learn more about NAT Gateway](#).
- [Learn how to use NAT Gateway with Azure App Service and Azure Functions](#).
- Review [Architectural approaches for networking in multitenant solutions](#).

Multitenancy and Azure Storage

3/10/2022 • 11 minutes to read • [Edit Online](#)

Azure Storage is a foundational service used in almost every solution. Multitenant solutions often use Azure Storage for blob, file, queue, and table storage. On this page, we describe some of the features of Azure Storage that are useful for multitenant solutions, and then we provide links to the guidance that can help you, when you're planning how you're going to use Azure Storage.

Features of Azure Storage that support multitenancy

Azure Storage includes many features that support multitenancy.

Shared access signatures

When you work with Azure Storage from a client application, it's important to consider whether client requests should be sent through another component that you control, like a content delivery network or API, or if the client should connect directly to your storage account. There might be good reasons to send requests through another component, including caching data at the edge of your network. However, in some situations, it's advantageous for client endpoints to connect directly to Azure Storage to download or upload data. This connection helps you improve the performance of your solution, especially when you work with large blobs or large numbers of files. It also reduces the load on your backend applications and servers, and it reduces the number of network hops. A [shared access signature](#) (SAS) enables you to securely provide your client applications with access to objects in Azure Storage.

Shared access signatures can be used to restrict the scope of operations that a client can perform, and the objects that they can perform operations against. For example, if you have a shared storage account for all of your tenants, and you store all of tenant A's data in a blob container named `tenantA`, you can create an SAS that only permits tenant A's users to access that container. For more information, see [Isolation models](#) to explore the approaches you can use to isolate your tenants' data in a storage account.

The [Valet Key pattern](#) is useful as a way to issue constrained and scoped shared access signatures from your application tier. For example, suppose you have a multitenant application that allows users to upload videos. Your API or application tier can authenticate the client using your own authentication system. You can then provide a SAS to the client that allows them to upload a video file to a specified blob, into a container and blob path that you specify. The client then uploads the file directly to the storage account, avoiding the extra bandwidth and load on your API. If they try to read data from the blob container, or if they try to write data to a different part of the container to another container in the storage account, Azure Storage blocks the request. The signature expires after a configurable time period.

[Stored access policies](#) extend the SAS functionality, which enables you to define a single policy that can be used when issuing multiple shared access signatures.

Identity-based access control

Azure Storage also provides [identity-based access control](#) by using Azure Active Directory (Azure AD). This capability also enables you to use [attribute-based access control](#), which gives you finer-grained access to blob paths, or to blobs that have been tagged with a specific tenant ID.

Lifecycle management

When you use blob storage in a multitenant solution, your tenants might require different policies for data retention. When you store large volumes of data, you might also want to configure the data for a specific tenant to automatically be moved to the [cool or archive storage tiers](#), for cost-optimization purposes.

Consider using [lifecycle management policies](#) to set the blob lifecycle for all tenants, or for a subset of tenants. A lifecycle management policy can be applied to blob containers, or to a subset of blobs within a container. However, there are limits on the number of rules you can specify in a lifecycle management policy. Make sure you plan and test your use of this feature in a multitenant environment, and consider deploying multiple storage accounts, if you will exceed the limits.

Immutable storage

When you configure [immutable blob storage](#) on storage containers with [time-based retention policies](#), Azure Storage prevents deletion or modification of the data before a specified time. The prevention is enforced at the storage account layer and applies to all users. Even your organization's administrators can't delete immutable data.

Immutable storage can be useful when you work with tenants that have legal or compliance requirements to maintain data or records. However, you should consider how this feature is used within the context of your [tenant lifecycle](#). For example, if tenants are offboarded and request the deletion of their data, you might not be able to fulfill their requests. If you use immutable storage for your tenants' data, consider how you address this issue in your terms of service.

Server-side copy

In a multitenant system, there is sometimes a need to move data from one storage account to another. For example, if you move a tenant between deployment stamps or rebalance a [sharded](#) set of storage accounts, you need to copy or move a specific tenant's data. When working with large volumes of data, it's advisable to use [server-side copy APIs](#) to decrease the time it takes to migrate the data.

The [AzCopy tool](#) is an application that you can run from your own computer, or from a virtual machine, to manage the copy process. AzCopy is compatible with the server-side copy feature, and it provides a scriptable command-line interface that you can run from your own solutions. AzCopy is also helpful for uploading and downloading large volumes of data.

If you need to use the server-side copy APIs directly from your code, consider using the [Put Block From URL API](#), [Put Page From URL API](#), [Append Block From URL API](#), and the [Copy Blob From URL API](#) when working with smaller blobs.

Object replication

The [Object replication](#) feature automatically replicates data between a source and destination storage account. Object replication is asynchronous. In a multitenant solution, this feature can be useful when you need to continuously replicate data between deployment stamps, or in an implementation of the [Geode pattern](#).

Encryption

Azure Storage enables you to [provide encryption keys](#) for your data. In a multitenant solution, consider combining this capability with [encryption scopes](#), which enable you to define different encryption keys for different tenants, even if their data is stored in the same storage account. By using these features together, you can also provide tenants with control over their own data. If they need to deactivate their account, they can delete the encryption key and their data is no longer accessible.

Monitoring

When working with a multitenant solution, consider whether you need to [measure the consumption for each tenant](#), and define the specific metrics you need to track, such as the amount of storage used for each tenant (the capacity), or the number of operations performed for each tenant's data.

Azure Storage provides [built-in monitoring capabilities](#). It's important to consider the services you'll use within the Azure Storage account. For example, when you work with [blobs](#), it's possible to view the total capacity of a storage account, but not a single container. In contrast, when you work with file shares, it's possible to see the capacity for each share, but not for each folder.

You can also [log all of the requests made to Azure Storage](#), and then you can aggregate and analyze those logs.

This provides more flexibility in how you aggregate and group data for each tenant. However, in solutions that create high volumes of requests to Azure Storage, it's important to consider whether the benefit you gain from this approach justifies the cost involved in capturing and processing those logs.

[Azure Storage inventory](#) provides another approach to measure the total size of a blob container.

Isolation models

When working with a multitenant system using Azure Storage, you need to make a decision about the level of isolation you want to use. Azure Storage supports several isolation models.

Storage accounts per tenant

The strongest level of isolation is to deploy a dedicated storage account for a tenant. This ensures that all storage keys are isolated and can be rotated independently. This approach enables you to scale your solution to avoid limits and quotas that are applicable to each storage account, but you also need to consider the maximum number of storage accounts that can be deployed into a single Azure subscription.

NOTE

Azure Storage has many quotas and limits that you should consider when you select an isolation model. These include [Azure service limits](#), [scalability targets](#), and [scalability targets for the Azure Storage resource provider](#).

Additionally, each component of Azure Storage provides further options for tenant isolation.

Blob storage isolation models

Shared blob containers

When working with blob storage, you might choose to use a shared blob container, and you might then use blob paths to separate data for each tenant:

TENANT ID	EXAMPLE BLOB PATH
tenant-a	https://contoso.blob.core.windows.net/sharedcontainer/tenant-a/blob1.mp4
tenant-b	https://contoso.blob.core.windows.net/sharedcontainer/tenant-b/blob2.mp4

While this approach is simple to implement, in many scenarios, blob paths don't provide sufficient isolation across tenants. This is because blob storage doesn't typically provide a concept of directories or folders. This means you can't assign access to all blobs within a specified path. However, Azure Storage provides a capability to [list \(enumerate\) blobs that begin with a specified prefix](#), which can be helpful when you work with shared blob containers and don't require directory-level access control.

Azure Storage's [hierarchical namespace](#) feature provides the ability to have a stronger concept of a directory or folder, including directory-specific access control. This can be useful in some multitenant scenarios where you have shared blob containers, but you want to grant access to a single tenant's data.

In some multitenant solutions, you might only need to store a single blob or set of blobs for each tenant, such as tenant icons for customizing a user interface. In these scenarios, a single shared blob container might be sufficient. You could use the tenant identifier as the blob name, and then read a specific blob instead of enumerating a blob path.

When you work with shared containers, consider whether you need to track the data and Azure Storage service usage for each tenant, and plan an approach to do so. See [Monitoring](#) for further information.

Blob containers per tenant

You can create individual blob containers for each tenant within a single storage account. There is no limit to the number of blob containers that you can create, within a storage account.

By creating containers for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data. You can also easily monitor the capacity that each container uses.

File storage isolation models

Shared file shares

When working with file shares, you might choose to use a shared file share, and then you might use file paths to separate data for each tenant:

TENANT ID	EXAMPLE FILE PATH
tenant-a	https://contoso.file.core.windows.net/share/tenant-a/blob1.mp4
tenant-b	https://contoso.file.core.windows.net/share/tenant-b/blob2.mp4

When you use an application that can communicate using the Server Message Block (SMB) protocol, and when you use Active Directory Domain Services either on-premises or in Azure, file shares [support authorization](#) at both the share and the directory/file levels.

In other scenarios, consider using SAS to grant access to specific file shares or files. When you use SAS, you can't grant access to directories.

When you work with shared file shares, consider whether you need to track the data and Azure Storage service usage for each tenant, and then plan an approach to do so (as necessary). See [Monitoring](#) for further information.

File shares per tenant

You can create individual file shares for each tenant, within a single storage account. There is no limit to the number of file shares that you can create within a storage account.

By creating file shares for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data. You can also easily monitor the capacity each file share uses.

Table storage isolation models

Shared tables with partition keys per tenant

When using table storage with a single shared table, you can consider using the [built-in support for partitioning](#). Each entity must include a partition key. A tenant identifier is often a good choice for a partition key.

Shared access signatures and policies enable you to specify a partition key range, and Azure Storage ensures that requests containing the signature can only access the specified partition key ranges. This enables you to implement the [Valet Key pattern](#), which allows untrusted clients to access a single tenant's partition, without affecting other tenants.

For high-scale applications, consider the maximum throughput of each table partition and the storage account.

Tables per tenant

You can create individual tables for each tenant within a single storage account. There is no limit to the number of tables that you can create within a storage account.

By creating tables for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data.

Queue storage isolation models

Shared queues

If you choose to share a queue, consider the quotas and limits that apply. In solutions with a high request volume, consider whether the target throughput of 2,000 messages per second is sufficient.

Queues don't provide partitioning or subqueues, so data for all tenants could be intermingled.

Queues per tenant

You can create individual queues for each tenant within a single storage account. There is no limit to the number of queues that you can create within a storage account.

By creating queues for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data.

When you dynamically create queues for each tenant, consider how your application tier will consume the messages from each tenant's queue. For more advanced scenarios, consider using [Azure Service Bus](#), which supports features such as [topics and subscriptions](#), [sessions](#), and [message auto-forwarding](#), which can be useful in a multitenant solution.

Next steps

Review [storage and data approaches for multitenancy](#).

Multitenancy and Azure SQL Database

3/10/2022 • 2 minutes to read • [Edit Online](#)

Multitenant solutions on Azure commonly use Azure SQL Database. On this page, we describe some of the features of Azure SQL Database that are useful when working with multitenanted systems, and we link to guidance and examples for how to use Azure SQL in a multitenant solution.

Guidance

The Azure SQL Database team has published extensive guidance on implementing multitenant architectures with Azure SQL Database. See [Multi-tenant SaaS patterns with Azure SQL Database](#). Also, consider the guidance for [partitioning Azure SQL databases](#).

Features of Azure SQL Database that support multitenancy

Azure SQL Database includes a number of features that support multitenancy.

Elastic pools

Elastic pools enable you to share compute resources between a number of databases on the same server. By using elastic pools, you can achieve performance elasticity for each database, while also achieving cost efficiency by sharing your provisioned resources across databases. Elastic pools provide built-in protections against the [Noisy Neighbor problem](#).

More information:

- [SQL Database elastic pools](#)
- [Resource management in dense elastic pools](#)
- [Disaster recovery strategies for applications using SQL Database elastic pools](#)

Elastic database tools

The [Sharding pattern](#) enables you to scale your workload across multiple databases. Azure SQL Database provides tools to support sharding. These tools include the management of *shard maps* (a database that tracks the tenants assigned to each shard), as well as initiating and tracking queries and management operations on multiple shards by using *elastic jobs*.

More information:

- [Multi-tenant applications with elastic database tools and row-level security](#)
- [Scaling out with Azure SQL Database](#)
- [Elastic database jobs](#)
- The [Elastic Jobs tutorial](#) describes the process of creating, configuring, and managing elastic jobs.

Row-level security

Row-level security is useful for enforcing tenant-level isolation, when you use shared tables.

More information:

- [Video overview](#)
- [Documentation](#)
- [Multi-tenant applications with elastic database tools and row-level security](#)

Key management

The Always Encrypted feature provides the end-to-end encryption of your databases. If your tenants require they supply their own encryption keys, consider deploying separate databases for each tenant and consider enabling the Always Encrypted feature.

More information:

- [Always Encrypted](#)

Next steps

Review [storage and data approaches for multitenancy](#).

Related resources

- [Data partitioning strategies for Azure SQL Database](#)
- [Case study: Running 1M databases on Azure SQL for a large SaaS provider: Microsoft Dynamics 365 and Power Platform](#)
- **Sample:** The [Wingtip Tickets SaaS application](#) provides three multi-tenant examples of the same app; each explores a different database tenancy pattern on Azure SQL Database. The first uses a standalone application, per tenant with its own database. The second uses a multi-tenant app with a database, per tenant. The third sample uses a multi-tenant app with sharded multi-tenant databases.
- **Video:** [Multitenant design patterns for SaaS applications on Azure SQL Database](#)

Multitenancy and Azure Cosmos DB

3/10/2022 • 8 minutes to read • [Edit Online](#)

On this page, we describe some of the features of Azure Cosmos DB that are useful when working with multitenanted systems, and we link to guidance and examples for how to use Azure Cosmos DB in a multitenant solution.

Features of Azure Cosmos DB that support multitenancy

Partitioning

By using partitions with your Cosmos DB containers, you can create containers that are shared across multiple tenants. Typically you use the tenant identifier as a partition key, but you might also consider using multiple partition keys for a single tenant. A well-planned partitioning strategy effectively implements the [Sharding pattern](#). With large containers, Cosmos DB spreads your tenants across multiple physical nodes to achieve a high degree of scale.

More information:

- [Partitioning and horizontal scaling in Azure Cosmos DB](#)
- [Hierarchical partition keys](#)

Managing request units

Cosmos DB's pricing model is based on the number of *request units* per second that you provision or consume. A request unit is a logical abstraction of the cost of a database operation or query. Typically, you provision a defined number of request units per second for your workload, which is referred to as *throughput*. Cosmos DB provides several options for how you provision throughput. In a multitenant environment, the selection you make affects the performance and price of your Cosmos DB resources.

One tenancy model for Cosmos DB involves deploying separate containers for each tenant within a shared database. Cosmos DB enables you to provision request units for a database, and all of the containers share the request units. If your tenant workloads don't typically overlap, this can provide a useful approach to reduce your operational costs. However, note that this approach is susceptible to the [Noisy Neighbor problem](#) because a single tenant's container might consume a disproportionate amount of the shared provisioned request units. To mitigate this after you've identified noisy tenants, you can optionally set provisioned throughput on a specific container. The other containers in the database continue to share their throughput, but the noisy tenant consumes their own dedicated throughput.

Cosmos DB also provides a serverless tier, which is suited for workloads with intermittent or unpredictable traffic. Alternatively, autoscaling enables you to configure policies to specify the scaling of provisioned throughput. In a multitenant solution, you might combine all of these approaches to support different types of tenant.

NOTE

When planning your Cosmos DB configuration, ensure you consider the [service quotas and limits](#).

To monitor and manage the costs that are associated with each tenant, every operation using the Cosmos DB API includes the request units consumed. You can use this information to aggregate and compare the actual request units consumed by each tenant, and you can then identify tenants with different performance characteristics.

More information:

- [Provisioned throughput](#)
- [Autoscale](#)
- [Serverless](#)
- [Measuring the RU charge of a request](#)
- [Azure Cosmos DB service quotas](#)

Customer-managed keys

Some tenants might require the use of their own encryption keys. Cosmos DB provides a customer-managed key feature. This feature is applied at the level of a Cosmos DB account, so tenants who require their own encryption keys need to be deployed using dedicated Cosmos DB accounts.

More information:

- [Configure customer-managed keys for your Azure Cosmos account with Azure Key Vault](#)

Isolation models

When working with a multitenant system that uses Azure Cosmos DB, you need to make a decision about the level of isolation you want to use. Azure Cosmos DB supports several isolation models:

	SHARED CONTAINERS WITH PARTITION KEYS PER TENANT	CONTAINER WITH SHARED THROUGHPUT PER TENANT	CONTAINER WITH DEDICATED THROUGHPUT PER TENANT	DATABASE ACCOUNT PER TENANT
Isolation options	<ul style="list-style-type: none">● Share throughput across tenants grouped by container (great for lowering cost on 'spiky' tenants).● Enables easy queries across tenants (containers act as boundary for queries). Mitigate a noisy neighbor blast radius (group tenants by container).	<ul style="list-style-type: none">● Share throughput across tenants that are grouped by database (which is great for lowering costs on 'spiky' tenants).● Easy management of tenants (drop the container when the tenant leaves).● Mitigate noisy neighbor blast radius (group tenants by database).	<ul style="list-style-type: none">● Independent throughput options (the dedicated throughput eliminates noisy neighbors).● Group tenants within database account(s), based on regional needs.	<ul style="list-style-type: none">● Independent geo-replication knobs.● Multiple throughput options (the dedicated throughput eliminates noisy neighbors).
Throughput requirements	>0 RUs per tenant	>100 RUs per tenant	>400 RUs per tenant	>400 RUs per tenant
Example use case	B2C apps	Standard offer for B2B apps	Premium offer for B2B apps	Premium offer for B2B apps

Shared container with partition keys per tenant

When you use a single container for multiple tenants, you can make use of Cosmos DB's partitioning support. By using separate partition keys for each tenant, you can easily query the data for a single tenant. You can also query across multiple tenants, even if they are in separate partitions. However, [cross-partition queries](#) have a higher request unit (RU) cost than single-partition queries.

This approach tends to work well when the amount of data stored for each tenant is small. It can be a good choice for building a [pricing model](#) that includes a free tier, and for business-to-consumer (B2C) solutions. In general, by using shared containers, you achieve the highest density of tenants and therefore the lowest price per tenant.

It's important to consider the throughput of your container. All of the tenants will share the container's throughput, so the [Noisy Neighbor problem](#) can cause performance challenges if your tenants have high or overlapping workloads. This problem can sometimes be mitigated by grouping subsets of tenants into different containers, and by ensuring that each tenant group has compatible usage patterns. Alternatively, you can consider a hybrid multi- and single-tenant model, where smaller tenants are grouped into shared partitioned containers, and large customers have dedicated containers.

It's also important to consider the amount of data you store in each logical partition. Azure Cosmos DB allows each logical partition to grow to up to 20 GB. If you have a single tenant that needs to store more than 20 GB of data, consider spreading the data across multiple logical partitions. For example, instead of having a single partition key of `Contoso`, you might *salt* the partition keys by creating multiple partition keys for a tenant, such as `Contoso1`, `Contoso2`, and so forth. When you query the data for a tenant, you can use the `WHERE IN` clause to match all of the partition keys. [Hierarchical partition keys](#) can also be used to support large tenants.

Consider the operational aspects of your solution, and the different phases of the [tenant lifecycle](#). For example, when a tenant moves to a dedicated pricing tier, you will likely need to move the data to a different container. When a tenant is deprovisioned, you need to run a delete query on the container to remove the data, and for large tenants, this query might consume a significant amount of throughput while it executes.

Container per tenant

You can provision dedicated containers for each tenant. This can work well when the data you store for your tenant can be combined into a single container.

When using a container for each tenant, you can consider sharing throughput with other tenants by provisioning throughput at the database level. Consider the restrictions and limits around the [minimum number of request units for your database](#) and the [maximum number of containers in the database](#). Also, consider whether your tenants require a guaranteed level of performance, and whether they're susceptible to the [Noisy Neighbor problem](#). If necessary, plan to group tenants into different databases that are based on workload patterns.

Alternatively, you can provision dedicated throughput for each container. This works well for larger tenants, and for tenants that are at risk of the [Noisy Neighbor problem](#). However, the baseline throughput for each tenant is higher, so consider the minimum requirements and cost implications of this model.

Lifecycle management is generally simpler when containers are dedicated to tenants. You can [easily move tenants between shared and dedicated throughput models](#), and when deprovisioning a tenant, you can quickly delete the container.

Database account per tenant

Cosmos DB enables you to provision separate database accounts for each tenant, which provides the highest level of isolation, but the lowest density. A single database account is dedicated to a tenant, which means they are not subject to the noisy neighbor problem. You can also configure the location of the database account according to the tenant's requirements, and you can tune the configuration of Cosmos DB features, such as geo-replication and customer-managed encryption keys, to suit each tenant's requirements. When using a dedicated

Cosmos DB account per tenant, consider the [maximum number of Cosmos DB accounts per Azure subscription](#).

If you allow tenants to migrate from a shared account to a dedicated Cosmos DB account, consider the migration approach you'll use to move a tenant's data between the old and new accounts.

Hybrid approaches

You can consider a combination of the above approaches to suit different tenants' requirements and [your pricing model](#). For example:

- Provision all free trial customers within a shared container, and use the tenant ID or a [synthetic key partition key](#).
- Offer a paid *Bronze* tier that deploys a dedicated container per tenant, but with [shared throughput on a database](#).
- Offer a higher *Silver* tier that provisions dedicated throughput for the tenant's container.
- Offer the highest *Gold* tier, and provide a dedicated database account for the tenant, which also allows tenants to select the geography for their deployment.

Next steps

Review [storage and data approaches for multitenancy](#).

Related resources

- [Azure Cosmos DB and multitenant systems](#): A blog post that discusses how to build a multitenant system that uses Azure Cosmos DB.
- [Multitenant applications with Azure Cosmos DB](#) (video)
- [Building a multitenant SaaS with Azure Cosmos DB and Azure](#) (video): A real-world case study of how Whally, a multitenant SaaS startup, built a modern platform from scratch on Azure Cosmos DB and Azure. Whally shows the design and implementation decisions they made that relate to partitioning, data modeling, secure multitenancy, performance, real-time streaming from change feed to SignalR, and more, all using ASP.NET Core on Azure App Services.

Multitenancy and Azure Database for PostgreSQL

3/10/2022 • 2 minutes to read • [Edit Online](#)

Many multitenant solutions on Azure use the open-source relational database management system Azure Database for PostgreSQL. In this article, we review the features of Azure Database for PostgreSQL that are useful when working with multitenant systems. The article also links to guidance and examples for how to use Azure Database for PostgreSQL, in a multitenant solution.

Deployment modes

There are three deployment modes available for Azure Database for PostgreSQL that are suitable for use with multitenant applications:

- [Single Server](#) - The basic PostgreSQL service that has a broad set of supported features and [service limits](#).
- [Flexible Server](#) - Supports higher [service limits](#) and larger SKUs than single server. This is a good choice for most multitenant deployments that don't require the high scalability that's provided by Hyperscale (Citus).
- [Hyperscale \(Citus\)](#) - Azure managed database service designed for solutions requiring a high level of scale, which often includes multitenanted applications.

Features of Azure Database for PostgreSQL that support multitenancy

When you're building a multitenant application using Azure Database for PostgreSQL, there are a number of features that you can use to enhance the solution.

NOTE

Some features are only available in specific [deployment modes](#). These features are indicated in the guidance below.

Row-level security

Row-level security is useful for enforcing tenant-level isolation, when you use shared tables. In PostgreSQL, row-level security is implemented by applying *row security policies* to tables to restrict access to rows by tenant.

More information:

- [Row security policies in PostgreSQL](#)

Horizontal scaling with sharding

The [Sharding pattern](#) enables you to scale your workload across multiple databases or database servers.

Solutions that need a very high level of scale can use Azure Database for PostgreSQL Hyperscale (Citus). This deployment mode enables horizontal sharding of tenants across multiple servers (nodes). By using *distributed tables* in multitenant databases, you can ensure all data for a tenant is stored on the same node, which increases query performance.

More information:

- [Designing a multitenant database in Hyperscale \(Citus\)](#)
- [Distributed tables](#)
- Choosing a [distribution column](#) in a distributed table.
- A guide to using [Citus for multitenant applications](#).

Connection pooling

Postgres uses a process-based model for connections. This model makes it inefficient to maintain large numbers of idle connections. Some multitenant architectures require a large number of active connections, which will negatively impact the performance of the Postgres server.

Connection pooling via PgBouncer is installed by default in Azure Database for PostgreSQL [Flexible Server](#) and [Hyperscale \(Citus\)](#). Connection pooling via PgBouncer is not built-in to [Single Server](#), but it can be installed on a separate server.

More information:

- [PgBouncer in Azure Database for PostgreSQL - Flexible Server](#)
- [Azure Database for PostgreSQL – Hyperscale \(Citus\) connection pooling](#)
- [Steps to install and set up PgBouncer connection pooling proxy with Azure Database for PostgreSQL](#)

Next steps

Review [storage and data approaches for multitenancy](#).

Checklist for architecting and building multitenant solutions on Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

When you build your multitenant solution in Azure, there are many elements that you need to consider. Use this checklist as a starting point to help you design and build your multitenant solution. This checklist is a companion resource to the [Architecting multitenant solutions on Azure](#) series of articles. The checklist is structured around the business and technical considerations, and the five pillars of the [Azure Well-Architected Framework](#).

Business considerations

- Understand what kind of solution you're creating, such as business-to-business (B2B), business-to-consumer (B2C), or your enterprise software, and [how tenants are different from users](#).
- [Define your tenants](#). Understand how many tenants you'll support initially, and your growth plans.
- [Define your pricing model](#) and ensure it aligns with your [tenants' consumption of Azure resources](#).
- Understand whether you need to separate your tenants into different [tiers](#). Tiers might have different pricing, features, performance promises, geographic locations, and so forth.
- Based on your customers' requirements, decide on the [tenancy models](#) that are appropriate for various parts of your solution.
- When you're ready, sell your B2B multitenant solution using the [Microsoft Commercial Marketplace](#).

Reliability considerations

- Review the [Azure Well-Architected Reliability checklist](#), which is applicable to all workloads.
- Understand the [Noisy Neighbor antipattern](#). Prevent individual tenants from impacting the system's availability for other tenants.
- [Design your multitenant solution](#) for the level of growth that you expect. But don't overengineer for unrealistic growth.
- Define service-level objectives (SLOs) and optionally [service-level agreements \(SLAs\)](#) for your solution. SLAs and SLOs should be based on the requirements of your tenants, as well as the [composite SLA of the Azure resources in your architecture](#).
- Test the [scale](#) of your solution. Ensure that it performs well under all levels of load, and that it scales correctly as the number of tenants increases.
- Apply [chaos engineering principles](#) to test the reliability of your solution.

Security considerations

- Apply the [Zero Trust](#) and least privilege principles in all layers of your solution.
- Ensure that you can [correctly map user requests](#) to tenants. Consider including the tenant context as part of the identity system, or by using another means, like application-level tenant authorization.
- Design for [tenant isolation](#). Continuously [test your isolation model](#).
- Ensure that your application code prevents any cross-tenant access or data leakage.
- Perform ongoing penetration testing and security code reviews.
- Understand your tenants' [compliance requirements](#), including data residency and any compliance or regulatory standards that they require you to meet.

- Correctly [manage domain names](#) and avoid vulnerabilities like [dangling DNS](#) and [subdomain takeover attacks](#).
- Follow [service-specific guidance](#) for multitenancy.

Cost Optimization considerations

- Review the [Azure Well-Architected Cost Optimization checklist](#), which is applicable to all workloads.
- Ensure you can adequately [measure per-tenant consumption](#) and correlate it with [your infrastructure costs](#).
- Avoid [antipatterns](#). Antipatterns include failing to track costs, tracking costs with unnecessary precision, real-time measurement, and using monitoring tools for billing.

Operational Excellence considerations

- Review the [Azure Well-Architected Operational Excellence checklist](#), which is applicable to all workloads.
- Use automation to manage the [tenant lifecycle](#), such as onboarding, [deployment](#), [provisioning](#), and [configuration](#).
- Find the right balance for [deploying service updates](#). Consider both your tenants' requirements and your own operational requirements.
- Monitor the health of the overall system, as well as each tenant.
- Configure and test alerts to notify you when specific tenants are experiencing issues or are exceeding their consumption limits.
- [Organize your Azure resources](#) for isolation and scale.
- Avoid [deployment and configuration antipatterns](#). Antipatterns include running separate versions of the solution for each tenant, hardcoding tenant-specific configurations or logic, and manual deployments.

Performance Efficiency considerations

- Review the [Azure Well-Architected Performance Efficiency checklist](#), which is applicable to all workloads.
- If you use shared infrastructure, plan for how you'll mitigate [Noisy Neighbor](#) concerns. Ensure that one tenant can't reduce the performance of the system for other tenants.
- Determine how you'll scale your [compute](#), [storage](#), [networking](#), and other Azure resources to match the demands of your tenants.
- Consider each Azure resource's scale limits. [Organize your resources](#) appropriately, in order to avoid [resource organization antipatterns](#). For example, don't over-architect your solution to work within unrealistic scale requirements.

Next steps

- Review [architectural considerations for multitenant solutions](#).
- Review [architectural approaches for multitenancy](#).
- Review [service-specific guidance for multitenancy](#).
- Review additional [resources for architects and developers of multitenant solutions](#).

Resources for architects and developers of multitenant solutions

3/10/2022 • 7 minutes to read • [Edit Online](#)

Architectures for multitenant applications

The following articles provide examples of multitenant architectures on Azure.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Multitenant SaaS on Azure	Reference architecture for a multitenant SaaS scenario on Azure, which is deployed in multiple regions	Web
Use Application Gateway Ingress Controller with a multi-tenant Azure Kubernetes Service	Example for implementing multitenancy with AKS and AGIC	Kubernetes
Serverless batch processing with Durable Functions in Azure Container Instances	Use cases include multitenant scenarios, where some tenants need large computing power, while other tenants have small computing requirements	Containers
All multitenant architectures	Lists all the architectures that include multitenancy	Multiple

Cloud design patterns

The following [cloud design patterns](#) are frequently used in multitenant architectures.

PATTERN	SUMMARY
Deployment Stamps pattern	Deploy multiple independent copies (scale units) of application components, including data stores.
Federated Identity	Delegate authentication to an external identity provider.
Gatekeeper	Protect applications and services, by using a dedicated host instance that acts as a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them.
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes, in order to smooth intermittent heavy loads.
Sharding	Divide a data store into a set of horizontal partitions or shards.

PATTERN	SUMMARY
Throttling	Control the consumption of resources that are used by an instance of an application, an individual tenant, or an entire service.

Antipatterns

Consider the [Noisy Neighbor antipattern](#), in which the activity of one tenant can have a negative impact on another tenant's use of the system.

Microsoft Azure Well-Architected Framework

While the entirety of the [Azure Well-Architected Framework](#) is important for all solutions, pay special attention to the [Resiliency pillar](#). The nature of cloud hosting leads to applications that are often multitenant, use shared platform services, compete for resources and bandwidth, communicate over the internet, and run on commodity hardware. This increases the likelihood that both transient and more permanent faults will arise.

Multitenant architectural guidance

- [Accelerate and De-Risk Your Journey to SaaS](#) (video): This video provides guidance for transitioning to the "software as a service" (SaaS) delivery model - whether you're starting by lifting-and-shifting an existing solution from on-premises to Azure, considering a multitenant architecture, or looking to modernize an existing SaaS web application.

Resources for Azure services

Governance and compliance

- [Organizing and managing multiple Azure subscriptions](#): It's important to consider how you manage your Azure subscriptions, as well as how you allocate tenant resources to subscriptions.
- [Cross-tenant management experiences](#): As a service provider, you can use Azure Lighthouse to manage resources, for multiple customers from within your own Azure Active Directory (Azure AD) tenant. Many tasks and services can be performed across managed tenants, by using Azure delegated resource management.
- [Azure Managed Applications](#): In a managed application, the resources are deployed to a resource group that's managed by the publisher of the app. The resource group is present in the consumer's subscription, but an identity in the publisher's tenant has access to the resource group.

Compute

- [Best practices for cluster isolation in Azure Kubernetes Service](#): AKS provides flexibility in how you can run multitenant clusters and can isolate resources. To maximize your investment in Kubernetes, you must first understand and implement AKS multitenancy and isolation features. This best practices article focuses on isolation for cluster operators.
- [Best practices for cluster security and upgrades in Azure Kubernetes Service](#): As you manage clusters in Azure Kubernetes Service (AKS), workload and data security is a key consideration. When you run multitenant clusters using logical isolation, you especially need to secure resource and workload access.

Storage and data

- [Azure Cosmos DB and multitenant systems](#): A blog post discussing how to build a multitenant system that uses Azure Cosmos DB.
- [Azure Cosmos DB hierarchical partition keys \(private preview\)](#): A blog post announcing the private preview of hierarchical partition keys for the Azure Cosmos DB Core (SQL) API. With hierarchical partition keys, also

known as sub-partitioning, you can now natively partition your container with up to three levels of partition keys. This enables more optimal partitioning strategies for multitenant scenarios or workloads that would otherwise use synthetic partition keys.

- [Azure SQL Database multitenant SaaS database tenancy patterns](#): A set of articles describing various tenancy models that are available for a multitenant SaaS application, using Azure SQL Database.
- [Running 1 million databases on Azure SQL for a large SaaS provider: Microsoft Dynamics 365 and Power Platform](#): A blog post describing how Dynamics 365 team manages databases at scale.
- [Design a multitenant database by using Azure Database for PostgreSQL Hyperscale](#)
- [Horizontal, vertical, and functional data partitioning](#): In many large-scale and multitenant solutions, data is divided into partitions that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and optimize performance. It can also provide a mechanism for dividing data, by the usage pattern and by the tenant.
- [Data partitioning strategies by Azure service](#): This article describes some strategies for partitioning data in various Azure data stores.
- [Building multitenant applications with Azure Database for PostgreSQL Hyperscale Citus](#) (video)
- [Multitenant applications with Azure Cosmos DB](#) (video)
- [Building a multitenant SaaS with Azure Cosmos DB and Azure](#) (video): A real-world case study of how Whally, a multitenant SaaS startup, built a modern platform from scratch on Azure Cosmos DB and Azure. Whally shows the design and implementation decisions they made related to partitioning, data modeling, secure multitenancy, performance, real-time streaming from change feed to SignalR and more, all using ASP.NET Core on Azure App Services.
- [Multitenant design patterns for SaaS applications on Azure SQL Database](#) (video)

Messaging

- [Azure Event Grid domains](#): Azure Event Grid domains allow you to manage multitenant eventing architectures, at scale.

Identity

- [Tenancy in Azure Active Directory](#): When developing apps, developers can choose to configure their app to be either single-tenant or multitenant, during app registration, in Azure Active Directory.
- [Custom-branded identity solution with Azure AD B2C](#): Azure Active Directory B2C is a customer identity access management solution that is capable of supporting millions of users and billions of authentications per day.
- [Identity management in multitenant applications](#): This series of articles describes best practices for multitenancy, when using Azure AD for authentication and identity management.
- [Build a multi-tenant daemon with the Microsoft identity platform endpoint](#): This sample application shows how to use the [Microsoft identity platform](#) endpoint to access the data of Microsoft business customers in a long-running, non-interactive process. It uses the OAuth2 client credentials grant to acquire an access token, which it then uses to call the Microsoft Graph and access organizational data.
- [Authenticate and authorize multitenant apps using Azure Active Directory \(Azure AD\)](#): Learn how Azure Active Directory enables you to improve the functionality of cloud-native apps in multitenant scenarios.
- [Azure Architecture Walkthrough: Building a multi-tenant Azure Architecture for a B2C scenario](#): a walk through the architecture behind a multi-tenant mobile app with Azure Active Directory B2C and API Management.

Analytics

- [Multitenancy solutions with Power BI embedded analytics](#): When designing a multitenant application that contains Power BI Embedded, you must carefully choose the tenancy model that best fits your needs.

IoT

- [Multitenancy in IoT Hub Device Provisioning Service](#): A multitenant IoT solution will commonly assign tenant

devices, by using a group of IoT hubs that are scattered across regions.

AI/ML

- [Design patterns for multitenant SaaS applications and Azure Cognitive Search](#): This document discusses tenant isolation strategies for multitenant applications that are built with Azure Cognitive Search.

Community Content

Kubernetes

- [Three Tenancy Models For Kubernetes](#): Kubernetes clusters are typically used by several teams in an organization. This article explains three tenancy models for Kubernetes.
- [Understanding Kubernetes Multi Tenancy](#): Kubernetes is not a multi-tenant system out of the box. While it is possible to configure multi-tenancy, this can be challenging. This article explains Kubernetes multi-tenancy types.
- [Kubernetes Multi-Tenancy – A Best Practices Guide](#): Kubernetes multi-tenancy is a topic that more and more organizations are interested in as their Kubernetes usage spreads out. However, since Kubernetes is not a multi-tenant system per se, getting multi-tenancy right comes with some challenges. This article describes these challenges and how to overcome them as well as some useful tools for Kubernetes multi-tenancy.
- [Capsule: Kubernetes multi-tenancy made simple](#): Capsule helps to implement a multi-tenancy and policy-based environment in your Kubernetes cluster. It is not intended to be yet another PaaS, instead, it has been designed as a micro-services-based ecosystem with the minimalist approach, leveraging only on upstream Kubernetes.
- [Loft: Add Multi-Tenancy To Your Clusters](#): Loft provides lightweight Kubernetes extensions for multi-tenancy.

Azure and Power Platform scenarios

3/10/2022 • 9 minutes to read • [Edit Online](#)

Power Platform provides tools for analyzing data, building solutions, automating processes, and creating virtual agents. Power Platform includes these products:

- [Power BI](#). Enable your employees to generate data-driven insights.
- [Power Apps](#). Enable anyone to build custom apps.
- [Power Automate](#). Give everyone the ability to automate organizational processes.
- [Power Virtual Agents](#). Build chatbots to engage with your customers and employees—no coding required.

This article provides summaries of solutions and architectures that use Power Platform together with Azure services.

Anyone can be a developer with Power Platform. Check out this short video to learn more:

Apache®, Apache Ignite, Ignite, and the flame logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Solutions across Azure and Power Platform

The following articles provide detailed analysis of solutions that feature integration between Azure and Power Platform.

Power Platform (general)

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Citizen AI with Power Platform	Learn how to use Azure Machine Learning and Power Platform to quickly create a machine learning proof of concept and a production version.	AI
Custom business processes	Deploy portals that use Power Apps to automate manual or paper-based processes and provide rich user experiences. Use Power BI to generate reports.	Integration
Modern data warehouse for small and medium businesses	Use Azure Synapse Analytics, Azure SQL Database, and Azure Data Lake Storage to modernize legacy and on-premises data. This solution integrates easily with Power Platform.	Analytics

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Virtual health on Microsoft Cloud for Healthcare	Develop a virtual health solution by using Microsoft Cloud for Healthcare. This solution uses Power Apps to host a patient portal and store data, Power BI for reporting, and Power Automate to trigger notifications.	Web

Power Apps

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Custom business processes	Deploy portals that use Power Apps to automate manual or paper-based processes and provide rich user experiences.	Integration
CI/CD for Microsoft Power Platform	Learn how to create an Azure CI/CD pipeline to manage your Power Platform application lifecycle.	DevOps
Eventual consistency between multiple Power Apps instances	Handle dependent data in a resilient way in Power Apps.	Web
Line of business extension	Modernize legacy systems by automating processes. Schedule calculations, connect to third-party data sources or legacy systems, and process and share data. Power Apps retrieves the data, and Power BI provides reporting.	Integration
Web and mobile front ends	Accelerate development by using a visual designer. Use Azure Functions for low-latency processing and Power Apps and Power Automate for out-of-the-box connectors.	Integration

Power Automate

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Extract text from objects using Power Automate and AI Builder	Use AI Builder and Azure Form Recognizer in a Power Automate workflow to extract text from images. The text can be used for indexing and retrieval.	AI
Power Automate deployment at scale	Learn how to use a hub-and-spoke architectural model to deploy Power Automate parent and child flows.	Integration
Web and mobile front ends	Accelerate development by using a visual designer. Use Azure Functions for low-latency processing and Power Apps and Power Automate for out-of-the-box connectors.	Integration

Power BI

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Advanced analytics	Combine any data at any scale and then use custom machine learning to get near real-time data analytics on streaming services. Power BI provides querying and reporting.	Analytics
Azure Machine Learning architecture	Learn how to build, deploy, and manage high-quality models with Azure Machine Learning, a service for the end-to-end machine learning lifecycle. Power BI provides data visualization.	AI
Campaign optimization with Azure HDInsight Spark	Use Microsoft Machine Learning Server to build and deploy a machine learning model to maximize the purchase rate of leads that are targeted by a marketing campaign. Power BI provides summaries of the effectiveness of the campaign recommendations.	Databases
Campaign optimization with SQL Server	Use machine learning and SQL Server 2016 R Services to optimize when and how to contact potential customers. Power BI provides data visualization.	Databases
Clinical insights with Microsoft Cloud for Healthcare	Gather insights from clinical and medical data by using Microsoft Cloud for Healthcare. Power BI reports provide insights on healthcare metrics.	Web
Data analysis for regulated industries	Learn about an architecture that you can use for data analysis workloads in regulated industries. The architecture includes ETL/ELT and Power BI.	Analytics
Data governance with Profisee and Azure Purview	Integrate Profisee master data management with Azure Purview to build a foundation for data governance and management. Produce and deliver high-quality, trusted data. Power BI is used as an analytics tool.	Databases
Data management across Azure Data Lake with Azure Purview	Use Azure Purview to build a foundation for data governance and management that can produce and deliver high-quality, trusted data. Azure Purview connects natively with Power BI.	Analytics

Architecture	Summary	Technology Focus
Defect prevention with predictive maintenance	Predict failures before they happen with real-time assembly line data, Azure Machine Learning, and Azure Synapse Analytics. Power BI enables visualization of real-time assembly-line data from Stream Analytics, and the predicted failures and alerts from Azure SQL Data Warehouse.	AI
Deliver highly scalable customer service and ERP applications	Use Azure SQL, Azure Cosmos DB, and Power BI to deliver highly scalable customer service and enterprise resource planning (ERP) applications that work with structured and unstructured data.	Analytics
Demand forecasting for shipping and distribution	Use historical demand data and the Microsoft AI platform to train a demand forecasting model for shipping and distribution solutions. A Power BI dashboard displays the forecasts.	Analytics
Finance management apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to store critical data with improved security and get high-value analytics and insights over aggregated data. Power BI supports native connectivity with PostgreSQL to ingest data for analytics.	Databases
Finance management apps using Azure Database for MySQL	Use Azure Database for MySQL to store critical data with improved security and get high-value analytics and insights over aggregated data. Power BI provides analytics.	Databases
Forecast energy and power demand	Forecast spikes in demand for energy products and services by using Azure Machine Learning and Power BI.	AI
HIPAA-compliant and HITRUST-compliant health data AI	Store, manage, and analyze HIPAA-compliant and HITRUST-compliant health data and medical records with a high level of built-in security. Power BI provides data visualization.	Serverless
Intelligent apps using Azure Database for MySQL	Use Azure Database for MySQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on. Power BI provides visualization and data analysis.	Databases

Architecture	Summary	Technology Focus
Intelligent apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on. Power BI provides visualization and data analysis.	Databases
Interactive querying with HDInsight	Use Apache Hive Live Long and Process (LLAP) to perform fast, interactive SQL queries at scale over structured or unstructured data. Power BI provides visualization and data analysis.	Databases
IoT-connected light, power, and internet for emerging markets	Learn how energy provider Veriown uses solar-powered IoT devices with Azure services to provide clean, low-cost power, light, and internet service to remote customers. Power BI provides reporting.	IoT
IoT using Azure Cosmos DB	Scale instantly and elastically to accommodate diverse and unpredictable IoT workloads without sacrificing ingestion or query performance. Use Power BI to analyze warehoused data.	IoT
Line of business extension	Modernize legacy systems by automating processes. Schedule calculations, connect to third-party data sources or legacy systems, and process and share data by using Power BI.	Integration
Loan charge-off prediction with HDInsight Spark	Learn how lending institutions can use Azure HDInsight and machine learning to predict the likelihood of loans getting charged off. Power BI provides a visualization dashboard.	Databases
Loan charge-off prediction with SQL Server	Build and deploy a machine learning model that uses SQL Server 2016 R Services to predict whether a bank loan will soon need to be charged off. Power BI provides interactive reports.	Databases
Loan credit risk and default modeling	Learn how SQL Server 2016 R Services can help lenders issue fewer unprofitable loans by predicting borrower credit risk and default probability. Power BI provides a dashboard to help lenders make decisions based on the predictions.	Databases

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Loan credit risk with SQL Server	Learn how lending institutions can use the predictive analytics of SQL Server 2016 R Services to reduce the number of loans to borrowers most likely to default. Power BI provides a dashboard to help lenders make decisions based on the predictions.	Databases
Manage data across your Azure SQL estate with Azure Purview	Improve your organization's governance process by using Azure Purview in your Azure SQL estate. Azure Purview connects natively to Power BI.	Analytics
Master data management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations. Power BI helps you to generate insights from the data.	Databases
Master data management with Profisee and Azure Data Factory	Integrate Profisee master data management with Azure Data Factory to get high-quality data for Azure Synapse and other analytics applications. Power BI provides data analysis.	Databases
Medical data storage solutions	Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security. Power BI provides data analysis.	Storage
Modern analytics architecture with Azure Databricks	Create a modern analytics architecture that uses Azure Databricks, Azure Data Lake Storage, Power BI, and other Azure services. Unify data, analytics, and AI workloads at any scale.	Analytics
Optimize marketing with machine learning	Build a machine learning model with Azure Machine Learning, Azure Synapse Analytics, and Power BI that optimizes big data marketing campaigns.	AI
Population health management for healthcare	Use population health management to improve clinical and health outcomes and reduce costs. Track, monitor, and benchmark data by using this process. Power BI provides analytics.	AI

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Predict length of stay and patient flow	Predict capacity and patient flow for your healthcare facility so that you can enhance the quality of care and improve operational efficiency. Power BI provides a dashboard to help you make decisions based on the predictions.	AI
Predict the length of stay in hospitals	Predict length of stay for hospital admissions to enhance care quality and operational workload efficiency and reduce re-admissions. Power BI provides data visualization.	Analytics
Predictive insights with vehicle telematics	Learn how car dealerships, manufacturers, and insurance companies can use Azure to gain predictive insights on vehicle health and driving habits. Power BI provides data visualizations for reporting.	AI
Predictive marketing with machine learning	Optimize big data marketing campaigns by using Azure Machine Learning and HDInsight Spark clusters. Power BI provides a dashboard to help you make decisions based on the predictions.	AI
Product recommendations for retail using Azure	Aggregate customer data into complete profiles. Use advanced Azure Machine Learning models to provide predictive insights on simulated customers. Use Power BI for data visualization.	AI
Project 15 Open Platform	Use Internet of Things technologies with the Project 15 Open Platform to accelerate innovation in species tracking, ecosystem monitoring, and other areas. Power BI provides visualization.	IoT
Quality assurance	Build a quality assurance system that collects data and improves productivity by identifying potential problems in a manufacturing pipeline before they occur. Use Power BI to visualize real-time operational dashboards.	AI
Serverless computing solution for LOB apps	Build and run customer onboarding applications without managing or maintaining infrastructure. Improve developer productivity with this serverless architecture. Power BI is used to store customer information.	Serverless

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Use a demand forecasting model for price optimization	Predict future customer demand and optimize pricing by using big-data and advanced-analytics services from Azure. Use Power BI to monitor the results.	Analytics

Related resources

- [Browse all our Power Platform architectures](#)

Azure and Microsoft 365 scenarios

3/10/2022 • 3 minutes to read • [Edit Online](#)

Microsoft 365 is a suite of apps that help you stay connected and get things done. It includes these tools:

- [Word](#). Create documents and improve your writing with built-in intelligent features.
- [Excel](#). Simplify complex data and create easy-to-read spreadsheets.
- [PowerPoint](#). Easily create polished presentations.
- [Teams](#). Bring everyone together in one place to meet, chat, call, and collaborate.
- [Outlook](#). Manage your email, calendar, tasks, and contacts in one place.
- [OneDrive](#). Save, access, edit, and share files.
- [Exchange](#). Work smarter with business-class email and calendaring.
- [SharePoint](#). Share and manage content, knowledge, and applications to enhance teamwork, make information easy to find, and collaborate across your organization.
- [Access](#). Create database apps easily in formats that best serve your business.
- [Publisher](#). Create professional layouts and publish content in a way that suits your audience.
- [Intune](#). Secure, deploy, and manage all users, apps, and devices without disrupting your processes.

This article provides a summary of architectures and solutions that use Azure together with Microsoft 365.

Watch this short video to learn how Microsoft 365 apps and services can help your organization work, learn, connect, and create:

Solutions across Azure and Microsoft 365

The following articles provide detailed analysis of solutions that feature integration between Azure and Microsoft 365.

Microsoft 365 (general)

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Azure AD security for AWS	Learn how Azure Active Directory (Azure AD) can help secure and protect Amazon Web Services (AWS) identity management and account access. If you already use Azure AD for Microsoft 365, this solution is easy to deploy.	Identity
Defender for Cloud Apps and Microsoft Sentinel for AWS	Learn how Microsoft Defender for Cloud Apps and Microsoft Sentinel can help secure and protect AWS account access and environments. If you already use Azure AD for Microsoft 365, this solution is easy to deploy.	Security

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Manage Microsoft 365 tenant configuration with Azure DevOps	Learn how to manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps.	Web
Power Automate deployment at scale	Learn how to use a hub-and-spoke architectural model to deploy Power Automate parent and child flows. This architecture is for Power Automate workflows that replace SharePoint 2010 workflows, and for new SharePoint Online sites.	Integration
Virtual health on Microsoft Cloud for Healthcare	Learn how to develop a virtual health solution by using Microsoft Cloud for Healthcare. Microsoft Cloud for Healthcare brings together capabilities from Microsoft 365, Azure, and other technologies to help healthcare organizations create fast, efficient, and highly secure healthcare solutions.	Web

Excel

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Interactive price analytics	Use transaction history data to develop a pricing strategy that shows how the demand for your products responds to your prices.	Analytics

Exchange Online

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Enhanced-security hybrid messaging—client access	Use Azure AD Multi-Factor Authentication to enhance your security in a client access scenario that uses Exchange.	Hybrid
Enhanced-security hybrid messaging—mobile access	Use Azure AD Multi-Factor Authentication to enhance your security in a mobile access scenario that uses Exchange.	Hybrid
Enhanced-security hybrid messaging—web access	Use Azure AD Multi-Factor Authentication to enhance your security in a web access scenario that uses Exchange.	Hybrid

SharePoint

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Enterprise-scale disaster recovery	Learn about a large-enterprise architecture for SharePoint, Dynamics CRM, and Linux web servers that runs on an on-premises datacenter and fails over to Azure infrastructure.	Management/Governance
Highly available SharePoint farm	Deploy a highly available SharePoint farm that uses Azure AD, a SQL Server Always On instance, and SharePoint resources.	Web
Hybrid SharePoint farm with Microsoft 365	Deliver highly available intranet capability and share hybrid workloads with Microsoft 365 by using SharePoint servers, Azure AD, and SQL Server.	Web
SharePoint farm for development testing	Deploy a SharePoint farm for development testing. Use Azure AD, SQL Server, and SharePoint resources for this agile development architecture.	DevOps

Teams

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Governance of Teams guest users	Learn how to use Teams and Azure AD entitlement management to collaborate with other organizations while maintaining control over resource use.	Identity
Provide security for your Teams channel bot and web app behind a firewall	Provide security for the connection to a Teams channel bot's web app by using Azure Private Link and a private endpoint.	Security
Teacher-provisioned virtual labs in Azure	Learn how you can use Azure Lab Services to set up identical VMs from templates for use in training, customer demos, and software development. Students and stakeholders can access the VMs via Teams integration.	DevOps

Related resources

- [Browse our Microsoft 365 architectures](#)

Azure for AWS professionals

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article helps Amazon Web Services (AWS) experts understand the basics of Microsoft Azure accounts, platform, and services. It also covers key similarities and differences between the AWS and Azure platforms.

You'll learn:

- How accounts and resources are organized in Azure.
- How available solutions are structured in Azure.
- How the major Azure services differ from AWS services.

Azure and AWS built their capabilities independently over time, so that each has important implementation and design differences.

Azure for AWS overview

Like AWS, Microsoft Azure is built around a core set of compute, storage, database, and networking services. In many cases, both platforms offer a basic equivalence between the products and services they offer. Both AWS and Azure allow you to build highly available solutions based on Windows or Linux hosts. So, if your preference for development is using Linux distributions and OSS technologies, both platforms are capable of doing the job.

While the capabilities of both platforms are similar, the resources that provide those capabilities are often organized differently. Exact one-to-one relationships between the services required to build a solution are not always clear. In other cases, a particular service might be offered on one platform, but not the other. See [charts of comparable Azure and AWS services](#).

Services

For a listing of how services map between the platforms, see [AWS to Azure services comparison](#).

Not all Azure products and services are available in all regions. Consult the [Products by Region](#) page for more details. You can find the uptime guarantees and downtime credit policies for each Azure product or service on the [Service Level Agreements](#) page.

Components

A number of core components on Azure and AWS have similar functionality. To review the differences, visit the component page for the topic you're interested in:

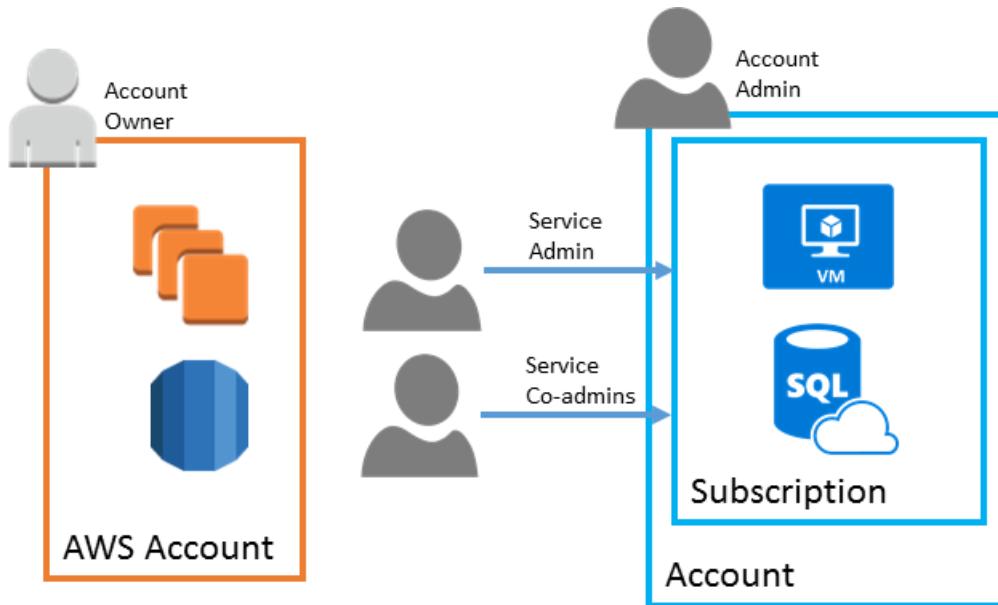
- [Accounts](#)
- [Compute](#)
- [Databases](#)
- [Messaging](#)
- [Networking](#)
- [Regions and Zones](#)
- [Resources](#)
- [Security & Identity](#)
- [Storage](#)

Azure and AWS accounts and subscriptions

3/10/2022 • 2 minutes to read • [Edit Online](#)

Azure services can be purchased using several pricing options, depending on your organization's size and needs. See the [pricing overview](#) page for details.

Azure subscriptions are a grouping of resources with an assigned owner responsible for billing and permissions management. Unlike AWS, where any resources created under the AWS account are tied to that account, subscriptions exist independently of their owner accounts, and can be reassigned to new owners as needed.



Comparison of structure and ownership of AWS accounts and Azure subscriptions

An Azure account represents a billing relationship and Azure subscriptions help you organize access to Azure resources. Account Administrator, Service Administrator, and Co-Administrator are the three classic subscription administrator roles in Azure:

- **Account Administrator.** The subscription owner and the billing owner for the resources used in the subscription. The account administrator can only be changed by transferring ownership of the subscription. Only one Account administrator is assigned per Azure Account.
- **Service Administrator.** This user has rights to create and manage resources in the subscription, but is not responsible for billing. By default, for a new subscription, the Account Administrator is also the Service Administrator. The account administrator can assign a separate user to the service administrator for managing the technical and operational aspects of a subscription. Only one service administrator is assigned per subscription.
- **Co-administrator.** There can be multiple co-administrators assigned to a subscription. Co-administrators have the same access privileges as the Service Administrator, but they cannot change the service administrator.

Below the subscription level user roles and individual permissions can also be assigned to specific resources, similarly to how permissions are granted to IAM users and groups in AWS. In Azure, all user accounts are associated with either a Microsoft Account or Organizational Account (an account managed through an Azure Active Directory).

Like AWS accounts, subscriptions have default service quotas and limits. For a full list of these limits, see [Azure subscription and service limits, quotas, and constraints](#). These limits can be increased up to the maximum by filing a support request in the management portal.

See also

- [Classic subscription administrator roles, Azure roles, and Azure AD roles](#)
- [How to add or change Azure administrator roles](#)
- [How to download your Azure billing invoice and daily usage data](#)

Compute services on Azure and AWS

3/10/2022 • 4 minutes to read • [Edit Online](#)

Amazon EC2 Instances and Azure virtual machines

Although AWS instance types and Azure virtual machine sizes are categorized similarly, the RAM, CPU, and storage capabilities differ.

- [Amazon EC2 Instance Types](#)
- [Sizes for virtual machines in Azure \(Windows\)](#)
- [Sizes for virtual machines in Azure \(Linux\)](#)

Similar to AWS' per second billing, Azure on-demand VMs are billed per second.

Amazon EBS and Azure Storage for VM disks

Durable data storage for Azure VMs is provided by [data disks](#) residing in blob storage. This is similar to how EC2 instances store disk volumes on Elastic Block Store (EBS). [Azure temporary storage](#) also provides VMs with the same low-latency temporary read-write storage as EC2 Instance Storage (also called ephemeral storage). [Azure Files](#) provides the VMs with the same functionality as [Amazon EFS](#).

Higher performance disk I/O is supported using [Azure premium storage](#). This is similar to the Provisioned IOPS storage options provided by AWS.

AWS Lambda, Azure Functions, Azure Web-Jobs, and Azure Logic Apps

[Azure Functions](#) is the primary equivalent of AWS Lambda in providing serverless, on-demand code. However, AWS Lambda functionality also overlaps with other Azure services:

- [WebJobs](#) allow you to create scheduled or continuously running background tasks.

Autoscaling, Azure VM scaling, and Azure App Service Autoscale

Autoscaling in Azure is handled by two services:

- [Virtual machine scale sets](#) allow you to deploy and manage an identical set of VMs. The number of instances can autoscale based on performance needs.
- [App Service Autoscale](#) provides the capability to autoscale Azure App Service solutions.

Container Service

The [Azure Kubernetes Service](#) supports Docker containers managed through Kubernetes. See [Container runtime configuration](#) for specifics on the hosting environment.

Distributed Systems Platform

[Service Fabric](#) is a platform for developing and hosting scalable [microservices-based](#) solutions.

Batch Processing

Azure Batch allows you to manage compute-intensive work across a scalable collection of virtual machines.

Service Comparison

Virtual servers

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Compute Cloud (EC2) Instances	Virtual Machines	Virtual servers allow users to deploy, manage, and maintain OS and server software. Instance types provide combinations of CPU/RAM. Users pay for what they use with the flexibility to change sizes.
Batch	Batch	Run large-scale parallel and high-performance computing applications efficiently in the cloud.
Auto Scaling	Virtual Machine Scale Sets	Allows you to automatically change the number of VM instances. You set defined metric and thresholds that determine if the platform adds or removes instances.
VMware Cloud on AWS	Azure VMware Solution	Seamlessly move VMware-based workloads from your datacenter to Azure and integrate your VMware environment with Azure. Keep managing your existing environments with the same VMware tools you already know while you modernize your applications with Azure native services. Azure VMware Solution is a Microsoft service, verified by VMware, that runs on Azure infrastructure.
Parallel Cluster	CycleCloud	Create, manage, operate, and optimize HPC and big compute clusters of any scale

Containers and container orchestrators

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Container Service (ECS) Fargate	Container Instances	Azure Container Instances is the fastest and simplest way to run a container in Azure, without having to provision any virtual machines or adopt a higher-level orchestration service.
Elastic Container Registry	Container Registry	Allows customers to store Docker formatted images. Used to create all types of container deployments on Azure.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Kubernetes Service (EKS)	Kubernetes Service (AKS)	Deploy orchestrated containerized applications with Kubernetes. Simplify monitoring and cluster management through auto upgrades and a built-in operations console. See AKS solution journey .
App Mesh	Service Fabric Mesh	Fully managed service that enables developers to deploy microservices applications without managing virtual machines, storage, or networking.

Container architectures

- [\[REDACTED\]](#)

Baseline architecture on Azure Kubernetes Service (AKS)

- 07/20/2020
- 37 min read

Deploy a baseline infrastructure that deploys an AKS cluster with focus on security.

- [\[REDACTED\]](#)

Microservices architecture on Azure Kubernetes Service (AKS)

- 5/07/2020
- 17 min read

Deploy a microservices architecture on Azure Kubernetes Service (AKS)

- [\[REDACTED\]](#)

CI/CD pipeline for container-based workloads

- 7/05/2018
- 7 min read

Build a DevOps pipeline for a Node.js web app with Jenkins, Azure Container Registry, Azure Kubernetes Service, Cosmos DB, and Grafana.

[view all](#)

Serverless

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Lambda	Functions	Integrate systems and run backend processes in response to events or schedules without provisioning or managing servers.

Serverless architectures

- [\[REDACTED\]](#)

[Social App for Mobile and Web with Authentication](#)

- 12/16/2019
- 3 min read

View a detailed, step-by-step diagram depicting the build process and implementation of the mobile client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.

- 

[HIPAA and HITRUST compliant health data AI](#)

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.

- 

[Cross Cloud Scaling Architecture](#)

- 12/16/2019
- 1 min read

Learn how to improve cross cloud scalability with solution architecture that includes Azure Stack. A step-by-step flowchart details instructions for implementation.

See also

- [Create a Linux VM on Azure using the portal](#)
- [Azure Reference Architecture: Running a Linux VM on Azure](#)
- [Get started with Node.js web apps in Azure App Service](#)
- [Azure Reference Architecture: Basic web application](#)
- [Create your first Azure Function](#)

Relational database technologies on Azure and AWS

3/10/2022 • 2 minutes to read • [Edit Online](#)

RDS and Azure relational database services

Azure provides several different relational database services that are the equivalent of AWS' Relational Database Service (RDS). These include:

- [SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [Azure Database for MariaDB](#)

Other database engines such as [SQL Server](#), [Oracle](#), and [MySQL](#) can be deployed using Azure VM Instances.

Costs for AWS RDS are determined by the amount of hardware resources that your instance uses, like CPU, RAM, storage, and network bandwidth. In the Azure database services, cost depends on your database size, concurrent connections, and throughput levels.

See also

- [Azure SQL Database Tutorials](#)
- [Azure SQL Managed Instance](#)
- [Configure geo-replication for Azure SQL Database with the Azure portal](#)
- [Introduction to Cosmos DB: A NoSQL JSON Database](#)
- [How to use Azure Table storage from Node.js](#)

Analytics and big data

Azure provides a package of products and services designed to capture, organize, analyze, and visualize large amounts of data consisting of the following services:

- [HDInsight](#): managed Apache distribution that includes Hadoop, Spark, Storm, or HBase.
- [Data Factory](#): provides data orchestration and data pipeline functionality.
- [Azure Synapse Analytics](#): an enterprise analytics service that accelerates time to insight, across data warehouses and big data systems.
- [Azure Databricks](#): a unified analytics platform for data analysts, data engineers, data scientists, and machine learning engineers.
- [Data Lake Store](#): analytics service that brings together enterprise data warehousing and big data analytics. Query data on your terms, using either serverless or dedicated resources—at scale.
- [Machine Learning](#): used to build and apply predictive analytics on data.
- [Stream Analytics](#): real-time data analysis.
- [Data Lake Analytics](#): large-scale analytics service optimized to work with Data Lake Store

- [Power BI](#): a business analytics service that provides the capabilities to create rich interactive data visualizations.

Service comparison

Type	AWS Service	Azure Service	Description
Relational database	RDS	SQL Database Database for MySQL Database for PostgreSQL Database for MariaDB	Managed relational database services in which resiliency, scale and maintenance are primarily handled by the Azure platform.
Serverless relational database	Amazon Aurora Serverless	Azure SQL Database serverless Serverless SQL pool in Azure Synapse Analytics	Database offerings that automatically scales compute based on the workload demand. You're billed per second for the actual compute used (Azure SQL)/data that's processed by your queries (Azure Synapse Analytics Serverless).
NoSQL/Document	DynamoDB SimpleDB Amazon DocumentDB	Cosmos DB	Cosmos DB is a globally distributed, multi-model database that natively supports multiple data models including key-value pairs, documents, graphs and columnar.
Caching	ElastiCache	Cache for Redis	An in-memory-based, distributed caching service that provides a high-performance store typically used to offload nontransactional work from a database.
Database migration	Database Migration Service	Database Migration Service	A service that executes the migration of database schema and data from one database format to a specific database technology in the cloud.

Database architectures

- [Database architectures](#)

Gaming using Cosmos DB

- 12/16/2019
- 1 min read

Elastically scale your database to accommodate unpredictable bursts of traffic and deliver low-latency

multi-player experiences on a global scale.

- [\[REDACTED\]](#)

[Oracle Database Migration to Azure](#)

- 12/16/2019
- 2 min read

Oracle DB migrations can be accomplished in multiple ways. This architecture covers one of these options wherein Oracle Active Data Guard is used to migrate the Database.

- [\[REDACTED\]](#)

[Retail and e-commerce using Azure MySQL](#)

- 12/16/2019
- 1 min read

Build secure and scalable e-commerce solutions that meet the demands of both customers and business using Azure Database for MySQL.

[view all](#)

See also

- [Azure AI Gallery](#)
- [Cloud-scale analytics](#)
- [Big data architecture style](#)
- [Azure Data Lake and Azure HDInsight blog](#)

Messaging services on Azure and AWS

3/10/2022 • 2 minutes to read • [Edit Online](#)

Simple Email Service

AWS provides the Simple Email Service (SES) for sending notification, transactional, or marketing emails. In Azure, third-party solutions, like [SendGrid](#), provide email services that can be incorporated into solutions to cater for various use cases.

Simple Queueing Service

AWS Simple Queueing Service (SQS) provides a messaging system for connecting applications, services, and devices within the AWS platform. Azure has two services that provide similar functionality:

- [Queue storage](#) is a cloud messaging service that allows communication between application components within Azure.
- [Service Bus](#) is a robust messaging system for connecting applications, services, and devices. By using the related [Service Bus relay](#), Service Bus can also connect to remotely hosted applications and services.

Messaging components

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Simple Queue Service (SQS)	Queue Storage	Provides a managed message queueing service for communicating between decoupled application components.
Simple Notification Service (SNS)	Service Bus	Supports a set of cloud-based, message-oriented middleware technologies, including reliable message queuing and durable publish/subscribe messaging.
Amazon EventBridge	Event Grid	A fully managed event routing service that allows for uniform event consumption using a publish/subscribe model.
Amazon Kinesis	Event Hubs	A fully managed, real-time data ingestion service. Stream millions of events per second, from any source, to build dynamic data pipelines and to immediately respond to business challenges.
Amazon MQ	Service Bus	Service Bus Premium is fully compliant with the Java/Jakarta EE Java Message Service (JMS) 2.0 API. Service Bus Standard supports the JMS 1.1 subset focused on queues.

Messaging architectures

-

Anomaly Detector Process

- 12/16/2019
- 1 min read

Learn more about Anomaly Detector with a step-by-step flowchart that details the process. See how anomaly detection models are selected with time-series data.

-

Scalable web application

- 10/03/2019
- 7 min read

Use the proven practices in this reference architecture to improve scalability and performance in an Azure App Service web application..

-

Enterprise integration using queues and events

- 12/03/2018
- 5 min read

Recommended architecture for implementing an enterprise integration pattern with Azure Logic Apps, Azure API Management, Azure Service Bus, and Azure Event Grid.

-

Ops automation using Event Grid

- 12/16/2019
- 1 min read

Event Grid allows you to speed automation and simplify policy enforcement. For example, Event Grid can notify Azure Automation when a virtual machine is created, or a SQL Database is spun up. These events can be used to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, or file work items.

Networking on Azure and AWS

3/10/2022 • 5 minutes to read • [Edit Online](#)

Elastic Load Balancing, Azure Load Balancer, and Azure Application Gateway

The Azure equivalent of the Elastic Load Balancing services are:

- [Load Balancer](#): Provides the same network layer 4 capabilities as the AWS Network Load Balancer and Classic Load Balancer, allowing you to distribute traffic for multiple VMs at the network level. It also provides a failover capability.
- [Application Gateway](#): Offers application-level rule-based routing comparable to the AWS Application Load Balancer.

Route 53, Azure DNS, and Azure Traffic Manager

In AWS, Route 53 provides both DNS name management and DNS-level traffic routing and failover services. In Azure this is handled through two services:

- [Azure DNS](#) provides domain and DNS management.
- [Traffic Manager](#) provides DNS level traffic routing, load balancing, and failover capabilities.

Direct connect and Azure ExpressRoute

Azure provides similar site-to-site dedicated connections through its [ExpressRoute](#) service. ExpressRoute allows you to connect your local network directly to Azure resources using a dedicated private network connection. Azure also offers more conventional [site-to-site VPN connections](#) at a lower cost.

Route tables

AWS provides route tables that contain routes to direct traffic, from a subnet/gateway subnet to the destination. In Azure, this feature is called user-defined routes.

With [user-defined routes](#), you can create custom or user-defined (static) routes in Azure, to override Azure's default system routes, or to add more routes to a subnet's route table.

Private Link

Similar to AWS PrivateLink, [Azure Private Link](#) provides private connectivity from a virtual network to an Azure platform as a service (PaaS) solution, a customer-owned service, or a Microsoft partner service.

VPC Peering, Azure VNet Peering

In AWS, a VPC peering connection is a networking connection between two VPCs, which enables you to route traffic between them using private IPv4 addresses or IPv6 addresses.

[Azure virtual network \(VNet\) peering](#) enables you to seamlessly connect two or more Virtual Networks in Azure. The virtual networks appear as one for connectivity purposes. The traffic between virtual machines in peered virtual networks uses the Microsoft backbone infrastructure. Like traffic between virtual machines in the same network, traffic is routed through Microsoft's private network only.

Content delivery networks - CloudFront and Azure CDN

In AWS, CloudFront provides CDN services, to globally deliver data, videos, applications, and APIs. This is similar to Azure Content Delivery Network (CDN).

[Azure CDN](#) is a feature-inclusive content delivery network bundle that is capable of handling most CDN workloads. Customers can choose to use Azure Content Delivery Network Standard from Verizon or Akamai. For a full list of Azure CDN product offerings, see [What are the comparisons between Azure CDN product features?](#)

Network service comparison

AREA	AWS SERVICE	AZURE SERVICE	DESCRIPTION
Cloud virtual networking	Virtual Private Cloud (VPC)	Virtual Network	Provides an isolated, private environment in the cloud. Users have control over their virtual networking environment, including selection of their own IP address range, creation of subnets, and configuration of route tables and network gateways.
NAT gateways	NAT Gateways	Virtual Network NAT	A service that simplifies outbound-only Internet connectivity for virtual networks. When configured on a subnet, all outbound connectivity uses your specified static public IP addresses. Outbound connectivity is possible without a load balancer or public IP addresses directly attached to virtual machines.
Cross-premises connectivity	VPN Gateway	VPN Gateway	Connects Azure virtual networks to other Azure virtual networks, or customer on-premises networks (Site To Site). Allows end users to connect to Azure services through VPN tunneling (Point To Site).
DNS management	Route 53	DNS	Manage your DNS records using the same credentials and billing and support contract as your other Azure services

Area	AWS Service	Azure Service	Description
DNS-based routing	Route 53	Traffic Manager	A service that hosts domain names, plus routes users to Internet applications, connects user requests to datacenters, manages traffic to apps, and improves app availability with automatic failover.
Dedicated network	Direct Connect	ExpressRoute	Establishes a dedicated, private network connection from a location to the cloud provider (not over the Internet).
Load balancing	Network Load Balancer	Load Balancer	Azure Load Balancer load balances traffic at layer 4 (TCP or UDP). Standard Load Balancer also supports cross-region or global load balancing.
Application-level load balancing	Application Load Balancer	Application Gateway	Application Gateway is a layer 7 load balancer. It supports SSL termination, cookie-based session affinity, and round robin for load-balancing traffic.
Route table	Custom Route Tables	User Defined Routes	Custom, or user-defined (static) routes to override default system routes, or to add more routes to a subnet's route table.
Private link	PrivateLink	Azure Private Link	Azure Private Link provides private access to services that are hosted on the Azure platform. This keeps your data on the Microsoft network.
Private PaaS connectivity	VPC endpoints	Private Endpoint	Private Endpoint provides secured, private connectivity to various Azure platform as a service (PaaS) resources, over a backbone Microsoft private network.

Area	AWS Service	Azure Service	Description
Virtual network peering	VPC Peering	VNET Peering	VNet peering is a mechanism that connects two virtual networks (VNets) in the same region through the Azure backbone network. Once peered, the two virtual networks appear as one for all connectivity purposes.
Content delivery networks	Cloud Front	Azure CDN	The Azure Content Delivery Network is designed to send audio, video, apps, photos, and other files to your customers faster and more reliably, using the servers closest to each user. Acceleration Data Transfer provides dynamic site acceleration of non-cacheable, dynamic content that is generated by your web applications.
Network Monitoring	VPC Flow Logs	Azure Network Watcher	Azure Network Watcher allows you to monitor, diagnose, and analyze the traffic in Azure Virtual Network.

Networking architectures

- [Deploy highly available NVAs](#)

- 12/08/2018
 ● 7 min read

Learn how to deploy network virtual appliances for high availability in Azure. This article includes example architectures for ingress, egress, and both.

- [Hub-spoke network topology in Azure](#)

- 9/30/2020
 ● 7 min read

Learn how to implement a hub-spoke topology in Azure, where the hub is a virtual network and the spokes are virtual networks that peer with the hub.

- [Implement a secure hybrid network](#)

- 1/07/2020

- 9 min read

See a secure hybrid network that extends an on-premises network to Azure with a perimeter network between the on-premises network and an Azure virtual network.

[view all](#)

See also

- [Create a virtual network using the Azure portal](#)
- [Plan and design Azure Virtual Networks](#)
- [Azure Network Security Best Practices](#)

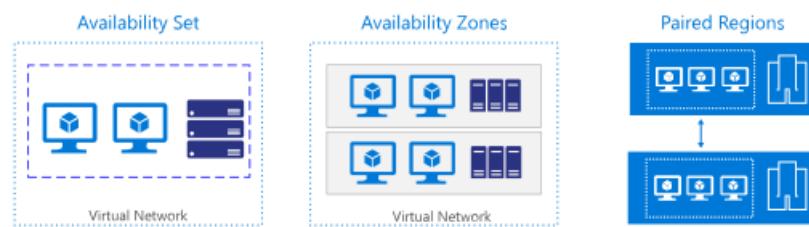
Regions and zones on Azure and AWS

3/10/2022 • 3 minutes to read • [Edit Online](#)

Failures can vary in the scope of their impact. Some hardware failures, such as a failed disk, may affect a single host machine. A failed network switch could affect a whole server rack. Less common are failures that disrupt a whole datacenter, such as loss of power in a datacenter. Rarely, an entire region could become unavailable.

One of the main ways to make an application resilient is through redundancy. But you need to plan for this redundancy when you design the application. Also, the level of redundancy that you need depends on your business requirements—not every application needs redundancy across regions to guard against a regional outage. In general, a tradeoff exists between greater redundancy and reliability versus higher cost and complexity.

In Azure, a region is divided into two or more Availability Zones. An Availability Zone corresponds with a physically isolated datacenter in the geographic region. Azure has numerous features for providing application redundancy at every level of potential failure, including **availability sets**, **availability zones**, and **paired regions**.



The diagram has three parts. The first part shows VMs in an availability set in a virtual network. The second part shows an availability zone with two availability sets in a virtual network. The third part shows regional pairs with resources in each region.

The following table summarizes each option.

	AVAILABILITY SET	AVAILABILITY ZONE	PAIRED REGION
Scope of failure	Rack	Datacenter	Region
Request routing	Load Balancer	Cross-zone Load Balancer	Traffic Manager
Network latency	Very low	Low	Mid to high
Virtual networking	VNet	VNet	Cross-region VNet peering

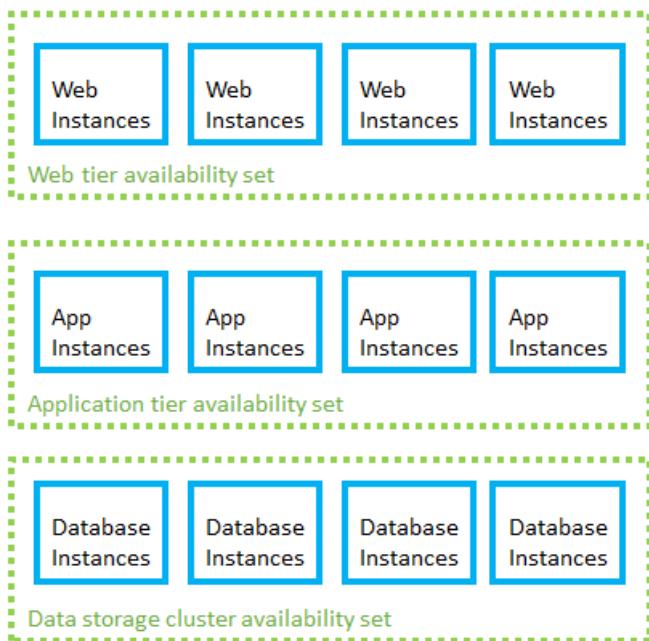
Availability sets

To protect against localized hardware failures, such as a disk or network switch failing, deploy two or more VMs in an availability set. An availability set consists of two or more *fault domains* that share a common power source and network switch. VMs in an availability set are distributed across the fault domains, so if a hardware failure affects one fault domain, network traffic can still be routed to the VMs in the other fault domains. For more information about Availability Sets, see [Manage the availability of Windows virtual machines in Azure](#).

When VM instances are added to availability sets, they are also assigned an **update domain**. An update domain is a group of VMs that are set for planned maintenance events at the same time. Distributing VMs across

multiple update domains ensures that planned update and patching events affect only a subset of these VMs at any given time.

Availability sets should be organized by the instance's role in your application to ensure one instance in each role is operational. For example, in a three-tier web application, create separate availability sets for the front-end, application, and data tiers.



Availability zones

An [Availability Zone](#) is a physically separate zone within an Azure region. Each Availability Zone has a distinct power source, network, and cooling. Deploying VMs across availability zones helps to protect an application against datacenter-wide failures.

Paired regions

To protect an application against a regional outage, you can deploy the application across multiple regions, using [Azure Traffic Manager](#) to distribute internet traffic to the different regions. Each Azure region is paired with another region. Together, these form a [regional pair](#). With the exception of Brazil South, regional pairs are located within the same geography in order to meet data residency requirements for tax and law enforcement jurisdiction purposes.

Unlike Availability Zones, which are physically separate datacenters but may be in relatively nearby geographic areas, paired regions are typically separated by at least 300 miles. This design ensures that large-scale disasters only affect one of the regions in the pair. Neighboring pairs can be set to sync database and storage service data, and are configured so that platform updates are rolled out to only one region in the pair at a time.

Azure [geo-redundant storage](#) is automatically backed up to the appropriate paired region. For all other resources, creating a fully redundant solution using paired regions means creating a full copy of your solution in both regions.

See also

- [Regions for virtual machines in Azure](#)
- [Availability options for virtual machines in Azure](#)
- [High availability for Azure applications](#)

- Failure and disaster recovery for Azure applications
- Planned maintenance for Linux virtual machines in Azure

Resource management on Azure and AWS

3/10/2022 • 2 minutes to read • [Edit Online](#)

The term "resource" in Azure is used in the same way as in AWS, meaning any compute instance, storage object, networking device, or other entity you can create or configure within the platform.

Azure resources are deployed and managed using one of two models: [Azure Resource Manager](#), or the older Azure [classic deployment model](#). Any new resources are created using the Resource Manager model.

Resource groups

Both Azure and AWS have entities called "resource groups" that organize resources such as VMs, storage, and virtual networking devices. However, [Azure resource groups](#) are not directly comparable to AWS resource groups.

While AWS allows a resource to be tagged into multiple resource groups, an Azure resource is always associated with one resource group. A resource created in one resource group can be moved to another group, but can only be in one resource group at a time. Resource groups are the fundamental grouping used by Azure Resource Manager.

Resources can also be organized using [tags](#). Tags are key-value pairs that allow you to group resources across your subscription irrespective of resource group membership.

Management interfaces

Azure offers several ways to manage your resources:

- [Web interface](#). Like the AWS Dashboard, the Azure portal provides a full web-based management interface for Azure resources.
- [REST API](#). The Azure Resource Manager REST API provides programmatic access to most of the features available in the Azure portal.
- [Command Line](#). The Azure CLI provides a command-line interface capable of creating and managing Azure resources. The Azure CLI is available for [Windows, Linux, and Mac OS](#).
- [PowerShell](#). The Azure modules for PowerShell allow you to execute automated management tasks using a script. PowerShell is available for [Windows, Linux, and Mac OS](#).
- [Templates](#). Azure Resource Manager templates provide similar JSON template-based resource management capabilities to the AWS CloudFormation service.

In each of these interfaces, the resource group is central to how Azure resources get created, deployed, or modified. This is similar to the role a "stack" plays in grouping AWS resources during CloudFormation deployments.

The syntax and structure of these interfaces are different from their AWS equivalents, but they provide comparable capabilities. In addition, many third-party management tools used on AWS, like [Hashicorp's Terraform](#) and [Netflix Spinnaker](#), are also available on Azure.

See also

- [Azure resource group guidelines](#)

Multi-cloud security and identity with Azure and Amazon Web Services (AWS)

3/10/2022 • 4 minutes to read • [Edit Online](#)

Many organizations are finding themselves with a de facto multi-cloud strategy, even if that wasn't their deliberate strategic intention. In a multi-cloud environment, it's critical to ensure consistent security and identity experiences to avoid increased friction for developers, business initiatives and increased organizational risk from cyberattacks taking advantage of security gaps.

Driving security and identity consistency across clouds should include:

- Multi-cloud identity integration
- Strong authentication and explicit trust validation
- Cloud Platform Security (multi-cloud)
- Microsoft Defender for Cloud
- Privilege Identity Management (Azure)
- Consistent end-to-end identity management

Multi-cloud identity integration

Customers using both Azure and AWS cloud platforms benefit from consolidating identity services between these two clouds using [Azure Active Directory](#) (Azure AD) and Single Sign-on (SSO) services. This model allows for a consolidated identity plane through which access to services in both clouds can be consistently accessed and governed.

This approach allows for the rich role-based access controls in Azure Active Directory to be enabled across the Identity & Access Management (IAM) services in AWS using rules to associate the user.userprincipalname and user.assignrole attributes from Azure AD into IAM permissions. This approach reduces the number of unique identities users and administrators are required to maintain across both clouds including a consolidation of the identity per account design that AWS employs. The [AWS IAM solution](#) allows for and specifically identifies Azure Active Directory as a federation and authentication source for their customers.

A complete walk-through of this integration can be found in the [Tutorial: Azure Active Directory single sign-on \(SSO\) integration with Amazon Web Services \(AWS\)](#).

Strong authentication and explicit trust validation

Because many customers continue to support a hybrid identity model for Active Directory services, it's increasingly important for security engineering teams to implement strong authentication solutions and block legacy authentication methods associated primarily with on-premises and legacy Microsoft technologies.

A combination of multi-factor authentication (MFA) and conditional access (CA) policies enable enhanced security for common authentication scenarios for end users in your organization. While MFA itself provides an increase level of security to confirm authentications, additional controls can be applied using [CA controls to block legacy authentication](#) to both Azure and AWS cloud environments. Strong authentication using only modern authentication clients is only possible with the combination of MFA and CA policies.

Cloud Platform Security (multi-cloud)

Once a common identity has been established in your multi-cloud environment, the [Cloud Platform Security](#)

(CPS) service of Microsoft Defender for Cloud Apps can be used to discover, monitor, assess, and protect those services. Using the Cloud Discovery dashboard, security operations personnel can review the apps and resources being used across AWS and Azure cloud platforms. Once services are reviewed and sanctioned for use, the services can then be managed as enterprise applications in Azure Active Directory to enable SAML, password-based, and linked Single Sign-On mode for the convenience of users.

CPS also provides for the ability to assess the cloud platforms connected for misconfigurations and compliance using vendor specific recommended security and configuration controls. This design enables organizations to maintain a single consolidated view of all cloud platform services and their compliance status.

CPS also provides access and session control policies to prevent and protect your environment from risky endpoints or users when data exfiltration or malicious files are introduced into those platforms.

Microsoft Defender for Cloud

[Microsoft Defender for Cloud](#) provides unified security management and threat protection across your hybrid and multi-cloud workloads, including workloads in Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). Defender for Cloud helps you find and fix security vulnerabilities, apply access and application controls to block malicious activity, detect threats using analytics and intelligence, and respond quickly when under attack.

To [protect your AWS-based resources on Microsoft Defender for Cloud](#), you can connect an account with either the Classic cloud connectors experience or the Environment settings page (in preview), which is recommended.

Privileged Identity Management (Azure)

To limit and control access for your highest privileged accounts in Azure AD, [Privileged Identity Management \(PIM\)](#) can be enabled to provide just-in-time access to services for Azure cloud services. Once deployed, PIM can be used to control and limit access using the assignment model for roles, eliminate persistent access for these privileged accounts, and provide additional discover and monitoring of users with these account types.

When combined with [Microsoft Sentinel](#), workbooks and playbooks can be established to monitor and raise alerts to your security operations center personnel when there is lateral movement of accounts that have been compromised.

Consistent end-to-end identity management

Ensure that all processes include an end-to-end view of all clouds as well as on-premises systems and that security and identity personnel are trained on these processes.

Using a single identity across Azure AD, AWS Accounts and on-premises services enable this end-to-end strategy and allows for greater security and protection of accounts for privileged and non-privileged accounts. Customers who are currently looking to reduce the burden of maintaining multiple identities in their multi-cloud strategy adopt Azure AD to provide consistent and strong control, auditing, and detection of anomalies and abuse of identities in their environment.

Continued growth of new capabilities across the Azure AD ecosystem helps you stay ahead of threats to your environment as a result of using identities as a common control plane in your multi-cloud environments.

Next steps

- [Azure Active Directory B2B](#): enables access to your corporate applications from partner-managed identities.
- [Azure Active Directory B2C](#): service offering support for single sign-on and user management for consumer-facing applications.
- [Azure Active Directory Domain Services](#): hosted domain controller service, allowing Active Directory

compatible domain join and user management functionality.

- [Getting started with Microsoft Azure security](#)
- [Azure Identity Management and access control security best practices](#)

Compare storage on Azure and AWS

3/10/2022 • 3 minutes to read • [Edit Online](#)

S3/EBS/EFS and Azure Storage

In the AWS platform, cloud storage is primarily broken down into three services:

- **Simple Storage Service (S3)**. Basic object storage that makes data available through an Internet accessible API.
- **Elastic Block Storage (EBS)**. Block level storage intended for access by a single VM.
- **Elastic File System (EFS)**. File storage meant for use as shared storage for up to thousands of EC2 instances.

In Azure Storage, subscription-bound [storage accounts](#) allow you to create and manage the following storage services:

- [Blob storage](#) stores any type of text or binary data, such as a document, media file, or application installer. You can set Blob storage for private access or share contents publicly to the Internet. Blob storage serves the same purpose as both AWS S3 and EBS.
- [Table storage](#) stores structured datasets. Table storage is a NoSQL key-attribute data store that allows for rapid development and fast access to large quantities of data. Similar to AWS' SimpleDB and DynamoDB services.
- [Queue storage](#) provides messaging for workflow processing and for communication between components of cloud services.
- [File storage](#) offers shared storage for legacy applications using the standard server message block (SMB) protocol. File storage is used in a similar manner to EFS in the AWS platform.

Glacier and Azure Storage

[Azure Archive Blob Storage](#) is comparable to AWS Glacier storage service. It is intended for rarely accessed data that is stored for at least 180 days and can tolerate several hours of retrieval latency.

For data that is infrequently accessed but must be available immediately when accessed, [Azure Cool Blob Storage tier](#) provides cheaper storage than standard blob storage. This storage tier is comparable to AWS S3 - Infrequent Access storage service.

Storage comparison

Object storage

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Simple Storage Services (S3)	Blob storage	Object storage service, for use cases including cloud applications, content distribution, backup, archiving, disaster recovery, and big data analytics.

Virtual server disks

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Block Store (EBS)	managed disks	SSD storage optimized for I/O intensive read/write operations. For use as high-performance Azure virtual machine storage.

Shared files

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic File System	Files	Provides a simple interface to create and configure file systems quickly, and share common files. Can be used with traditional protocols that access files over a network.

Archiving and backup

AWS SERVICE	AZURE SERVICE	DESCRIPTION
S3 Infrequent Access (IA)	Storage cool tier	Cool storage is a lower-cost tier for storing data that is infrequently accessed and long-lived.
S3 Glacier, Deep Archive	Storage archive access tier	Archive storage has the lowest storage cost and higher data retrieval costs compared to hot and cool storage.
Backup	Backup	Back up and recover files and folders from the cloud, and provide offsite protection against data loss.

Hybrid storage

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Storage Gateway	StorSimple	Integrates on-premises IT environments with cloud storage. Automates data management and storage, plus supports disaster recovery.
DataSync	File Sync	Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync.

Bulk data transfer

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Import/Export Disk	Import/Export	A data transport solution that uses secure disks and appliances to transfer large amounts of data. Also offers data protection during transit.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Import/Export Snowball, Snowball Edge, Snowmobile	Data Box	Petabyte- to exabyte-scale data transport solution that uses secure data storage devices to transfer large amounts of data to and from Azure.

Storage architectures

-

HIPAA and HITRUST compliant health data AI

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.

-

HPC Media Rendering

- 11/04/2020
- 2 min read

Optimize the media rendering process with a step-by-step HPC solution architecture from Azure that combines Azure CycleCloud and HPC Cache.

-

Medical Data Storage Solutions

- 12/16/2019
- 2 min read

Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security.

[view all](#)

See also

- [Microsoft Azure Storage Performance and Scalability Checklist](#)
- [Azure Storage security guide](#)
- [Best practices for using content delivery networks \(CDNs\)](#)

AWS to Azure services comparison

3/10/2022 • 28 minutes to read • [Edit Online](#)

This article helps you understand how Microsoft Azure services compare to Amazon Web Services (AWS). Whether you are planning a multicloud solution with Azure and AWS, or migrating to Azure, you can compare the IT capabilities of Azure and AWS services in all categories.

This article compares services that are roughly comparable. Not every AWS service or Azure service is listed, and not every matched service has exact feature-for-feature parity.

Azure and AWS for multicloud solutions

As the leading public cloud platforms, Azure and AWS each offer a broad and deep set of capabilities with global coverage. Yet many organizations choose to use both platforms together for greater choice and flexibility, as well as to spread their risk and dependencies with a multicloud approach. Consulting companies and software vendors might also build on and use both Azure and AWS, as these platforms represent most of the cloud market demand.

For an overview of Azure for AWS users, see [Introduction to Azure for AWS professionals](#).

Marketplace

AWS SERVICE	AZURE SERVICE	DESCRIPTION
AWS Marketplace	Azure Marketplace	Easy-to-deploy and automatically configured third-party applications, including single virtual machine or multiple virtual machine solutions.

AI and machine learning

AWS SERVICE	AZURE SERVICE	DESCRIPTION
SageMaker	Machine Learning	A cloud service to train, deploy, automate, and manage machine learning models.
Alexa Skills Kit	Bot Framework	Build and connect intelligent bots that interact with your users using text/SMS, Skype, Teams, Slack, Microsoft 365 mail, Twitter, and other popular services.
Lex	Speech Services	API capable of converting speech to text, understanding intent, and converting text back to speech for natural responsiveness.
Lex	Language Understanding (LUIS)	Allows your applications to understand user commands contextually.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Polly, Transcribe	Speech Services	Enables both Speech to Text, and Text into Speech capabilities.
Rekognition	Cognitive Services	Computer Vision: Extract information from images to categorize and process visual data. Face: Detect, identify, and analyze faces and facial expressions in photos.
Skills Kit	Virtual Assistant	The Virtual Assistant Template brings together a number of best practices we've identified through the building of conversational experiences and automates integration of components that we've found to be highly beneficial to Bot Framework developers.

AI and machine learning architectures

- [\[REDACTED\]](#)

[Image classification on Azure](#)

- 7/05/2018
- 4 min read

Learn how to build image processing into your applications by using Azure services such as the Computer Vision API and Azure Functions.

- [\[REDACTED\]](#)

[Predictive Marketing with Machine Learning](#)

- 12/16/2019
- 2 min read

Learn how to build a machine-learning model with Microsoft R Server on Azure HDInsight Spark clusters to recommend actions to maximize the purchase rate.

- [\[REDACTED\]](#)

[Scalable personalization on Azure](#)

- 5/31/2019
- 6 min read

Use machine learning to automate content-based personalization for customers.

[view all](#)

Big data and analytics

Data warehouse

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Redshift	Synapse Analytics	Cloud-based enterprise data warehouse (EDW) that uses massively parallel processing (MPP) to quickly run complex queries across petabytes of data.
Lake Formation	Data Share	A simple and safe service for sharing big data.

Data warehouse architectures

- [Data warehouse architecture](#)

Modern Data Warehouse Architecture

- 12/16/2019
- 2 min read

Explore a cloud data warehouse that uses big data. Modern data warehouse brings together all your data and scales easily as your data grows.

- [Modern Data Warehouse Architecture](#)

Automated enterprise BI

- 6/03/2020
- 13 min read

Automate an extract, load, and transform (ELT) workflow in Azure using Azure Data Factory with Azure Synapse Analytics.

[view all](#)

Time series

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Amazon Timestream	Azure Data Explorer Azure Time Series Insights	Fully managed, low latency, and distributed big data analytics platform that runs complex queries across petabytes of data. Highly optimized for log and time series data. Open and scalable end-to-end IoT analytics service. Collect, process, store, query, and visualize data at Internet of Things (IoT) scale--data that's highly contextualized and optimized for time series.

Time series architectures

- [Time series architecture](#)

IoT analytics with Azure Data Explorer

- 8/11/2020

- 3 min read

IoT Telemetry Analytics with Azure Data Explorer demonstrates near real-time analytics over fast flowing, high volume, wide variety of streaming data from IoT devices.

- [redacted]

[Azure Data Explorer interactive analytics](#)

- 8/11/2020
- 3 min read

Interactive Analytics with Azure Data Explorer focuses on its integration with the rest of the data platform ecosystem.

Big data processing

AWS SERVICE	AZURE SERVICE	DESCRIPTION
EMR	Azure Data Explorer	Fully managed, low latency, distributed big data analytics platform to run complex queries across petabytes of data.
EMR	Databricks	Apache Spark-based analytics platform.
EMR	HDInsight	Managed Hadoop service. Deploy and manage Hadoop clusters in Azure.
EMR	Data Lake Storage	Massively scalable, secure data lake functionality built on Azure Blob Storage.

Big data architectures

- [redacted]

[Azure data platform end-to-end](#)

- 1/31/2020
- 7 min read

Use Azure services to ingest, process, store, serve, and visualize data from different sources.

- [redacted]

[Campaign Optimization with Azure HDInsight Spark Clusters](#)

- 12/16/2019
- 4 min read

This solution demonstrates how to build and deploy a machine learning model with Microsoft R Server on Azure HDInsight Spark clusters to recommend actions to maximize the purchase rate of leads targeted by a campaign. This solution enables efficient handling of big data on Spark with Microsoft R Server.

- [redacted]

Big data analytics with Azure Data Explorer

- 8/11/2020
- 3 min read

Big Data Analytics with Azure Data Explorer demonstrates Azure Data Explorer's abilities to cater to volume, velocity, and variety of data, the three V's of big data.

[view all](#)

Data orchestration / ETL

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Data Pipeline, Glue	Data Factory	Processes and moves data between different compute and storage services, as well as on-premises data sources at specified intervals. Create, schedule, orchestrate, and manage data pipelines.
Glue	Azure Purview	A unified data governance service that helps you manage and govern your on-premises, multicloud, and software as a service (SaaS) data.
Dynamo DB	Table Storage, Cosmos DB	NoSQL key-value store for rapid development using massive semi-structured datasets.

Analytics and visualization

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Kinesis Analytics	Stream Analytics Azure Data Explorer Data Lake Analytics Data Lake Store	Storage and analysis platforms that create insights from large quantities of data, or data that originates from many sources.
QuickSight	Power BI	Business intelligence tools that build visualizations, perform ad hoc analysis, and develop business insights from data.
CloudSearch	Cognitive Search	Delivers full-text search and related search analytics and capabilities.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Athena	Data Lake Analytics Azure Synapse Analytics	<p>Provides a serverless interactive query service that uses standard SQL for analyzing databases.</p> <p>Azure Synapse Analytics is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics. It gives you the freedom to query data on your terms, using either serverless or dedicated resources at scale.</p>
Elasticsearch Service	Elastic on Azure	Use the Elastic Stack (Elastic, Logstash, and Kibana) to search, analyze, and visualize in real time.

Analytics architectures

- [\[REDACTED\]](#)

Advanced Analytics Architecture

- 12/16/2019
- 2 min read

Get near real-time data analytics on streaming services. This big data architecture allows you to combine any data at any scale with custom machine learning.

- [\[REDACTED\]](#)

Automated enterprise BI

- 6/03/2020
- 13 min read

Automate an extract, load, and transform (ELT) workflow in Azure using Azure Data Factory with Azure Synapse Analytics.

- [\[REDACTED\]](#)

Mass ingestion and analysis of news feeds on Azure

- 2/01/2019
- 5 min read

Create a pipeline for ingesting and analyzing text, images, sentiment, and other data from RSS news feeds using only Azure services, including Azure Cosmos DB and Azure Cognitive Services.

[view all](#)

Compute

Virtual servers

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Compute Cloud (EC2) Instances	Virtual Machines	Virtual servers allow users to deploy, manage, and maintain OS and server software. Instance types provide combinations of CPU/RAM. Users pay for what they use with the flexibility to change sizes.
Batch	Batch	Run large-scale parallel and high-performance computing applications efficiently in the cloud.
Auto Scaling	Virtual Machine Scale Sets	Allows you to automatically change the number of VM instances. You set defined metric and thresholds that determine if the platform adds or removes instances.
VMware Cloud on AWS	Azure VMware Solution	Seamlessly move VMware-based workloads from your datacenter to Azure and integrate your VMware environment with Azure. Keep managing your existing environments with the same VMware tools you already know while you modernize your applications with Azure native services. Azure VMware Solution is a Microsoft service, verified by VMware, that runs on Azure infrastructure.
Parallel Cluster	CycleCloud	Create, manage, operate, and optimize HPC and big compute clusters of any scale

Containers and container orchestrators

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Container Service (ECS) Fargate	Container Instances	Azure Container Instances is the fastest and simplest way to run a container in Azure, without having to provision any virtual machines or adopt a higher-level orchestration service.
Elastic Container Registry	Container Registry	Allows customers to store Docker formatted images. Used to create all types of container deployments on Azure.
Elastic Kubernetes Service (EKS)	Kubernetes Service (AKS)	Deploy orchestrated containerized applications with Kubernetes. Simplify monitoring and cluster management through auto upgrades and a built-in operations console. See AKS solution journey .

AWS SERVICE	AZURE SERVICE	DESCRIPTION
App Mesh	Service Fabric Mesh	Fully managed service that enables developers to deploy microservices applications without managing virtual machines, storage, or networking.

Container architectures

- [Containerized application on Azure Kubernetes Service \(AKS\)](#)

Baseline architecture on Azure Kubernetes Service (AKS)

- 07/20/2020
- 37 min read

Deploy a baseline infrastructure that deploys an AKS cluster with focus on security.

- [Baseline architecture on Azure Kubernetes Service \(AKS\)](#)

Microservices architecture on Azure Kubernetes Service (AKS)

- 5/07/2020
- 17 min read

Deploy a microservices architecture on Azure Kubernetes Service (AKS)

- [Microservices architecture on Azure Kubernetes Service \(AKS\)](#)

CI/CD pipeline for container-based workloads

- 7/05/2018
- 7 min read

Build a DevOps pipeline for a Node.js web app with Jenkins, Azure Container Registry, Azure Kubernetes Service, Cosmos DB, and Grafana.

[view all](#)

Serverless

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Lambda	Functions	Integrate systems and run backend processes in response to events or schedules without provisioning or managing servers.

Serverless architectures

- [Serverless architecture for a mobile and web application](#)

Social App for Mobile and Web with Authentication

- 12/16/2019
- 3 min read

View a detailed, step-by-step diagram depicting the build process and implementation of the mobile

client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.



HIPAA and HITRUST compliant health data AI

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.



Cross Cloud Scaling Architecture

- 12/16/2019
- 1 min read

Learn how to improve cross cloud scalability with solution architecture that includes Azure Stack. A step-by-step flowchart details instructions for implementation.

Database

TYPE	AWS SERVICE	AZURE SERVICE	DESCRIPTION
Relational database	RDS	SQL Database Database for MySQL Database for PostgreSQL Database for MariaDB	Managed relational database services in which resiliency, scale and maintenance are primarily handled by the Azure platform.
Serverless relational database	Amazon Aurora Serverless	Azure SQL Database serverless Serverless SQL pool in Azure Synapse Analytics	Database offerings that automatically scales compute based on the workload demand. You're billed per second for the actual compute used (Azure SQL)/data that's processed by your queries (Azure Synapse Analytics Serverless).
NoSQL/Document	DynamoDB SimpleDB Amazon DocumentDB	Cosmos DB	Cosmos DB is a globally distributed, multi-model database that natively supports multiple data models including key-value pairs, documents, graphs and columnar.

Type	AWS Service	Azure Service	Description
Caching	ElastiCache	Cache for Redis	An in-memory-based, distributed caching service that provides a high-performance store typically used to offload nontransactional work from a database.
Database migration	Database Migration Service	Database Migration Service	A service that executes the migration of database schema and data from one database format to a specific database technology in the cloud.

Database architectures

- [\[REDACTED\]](#)

Gaming using Cosmos DB

- 12/16/2019
- 1 min read

Elastically scale your database to accommodate unpredictable bursts of traffic and deliver low-latency multi-player experiences on a global scale.

- [\[REDACTED\]](#)

Oracle Database Migration to Azure

- 12/16/2019
- 2 min read

Oracle DB migrations can be accomplished in multiple ways. This architecture covers one of these options wherein Oracle Active Data Guard is used to migrate the Database.

- [\[REDACTED\]](#)

Retail and e-commerce using Azure MySQL

- 12/16/2019
- 1 min read

Build secure and scalable e-commerce solutions that meet the demands of both customers and business using Azure Database for MySQL.

[view all](#)

DevOps and application monitoring

AWS Service	Azure Service	Description
-------------	---------------	-------------

AWS SERVICE	AZURE SERVICE	DESCRIPTION
CloudWatch, X-Ray	Monitor	Comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.
CodeDeploy	DevOps	A cloud service for collaborating on code development.
CodeCommit		
CodePipeline		
Developer Tools	Developer Tools	Collection of tools for building, debugging, deploying, diagnosing, and managing multiplatform scalable apps and services.
CodeBuild	DevOps Pipeline	Fully managed build service that supports continuous integration and deployment.
	Github Actions	
Command Line Interface	CLI	Built on top of the native REST API across all cloud services, various programming language-specific wrappers provide easier ways to create solutions.
	PowerShell	
eksctl	az aks	Manage Azure Kubernetes Services.
AWS CloudShell	Azure Cloud Shell	Azure Cloud Shell is an interactive, authenticated, browser-accessible shell for managing Azure resources. It gives you the flexibility to choose the shell experience that best suits the way you work, either Bash or PowerShell.
OpsWorks (Chef-based)	Automation	Configures and operates applications of all shapes and sizes, and provides templates to create and manage a collection of resources.
CloudFormation	Resource Manager	Provides a way for users to automate the manual, long-running, error-prone, and frequently repeated IT tasks.
	Bicep	
	VM extensions	
	Azure Automation	

DevOps architectures

- Container CI/CD using Jenkins and Kubernetes on Azure Kubernetes Service (AKS)

- 12/16/2019
- 2 min read

Containers make it easy for you to continuously build and deploy applications. By orchestrating the deployment of those containers using Azure Kubernetes Service (AKS), you can achieve replicable, manageable clusters of containers.

- [Run a Jenkins server on Azure](#)

• 11/19/2020

• 6 min read

Recommended architecture that shows how to deploy and operate a scalable, enterprise-grade Jenkins server on Azure secured with single sign-on (SSO).

- [Run a Jenkins server on Azure](#)

DevOps in a hybrid environment

• 12/16/2019

• 3 min read

The tools provided in Azure allow for the implementation of a DevOps strategy that capably manages both cloud and on-premises environments in tandem.

[view all](#)

Internet of things (IoT)

AWS SERVICE	AZURE SERVICE	DESCRIPTION
IoT Core	IoT Hub	A cloud gateway for managing bidirectional communication with billions of IoT devices, securely and at scale.
Greengrass	IoT Edge	Deploy cloud intelligence directly onto IoT devices, catering to on-premises scenarios.
Kinesis Firehose, Kinesis Streams	Event Hubs	Services that facilitate the mass ingestion of events (messages), typically from devices and sensors. The data can then be processed in real-time micro-batches or be written to storage for further analysis.
IoT Things Graph	Digital Twins	Services you can use to create digital representations of real-world things, places, business processes, and people. Use these services to gain insights, drive the creation of better products and new customer experiences, and optimize operations and costs.

IOT architectures

- [Run a Jenkins server on Azure](#)

IoT Architecture Azure IoT Subsystems

- 12/16/2019
- 1 min read

Learn about our recommended IoT application architecture that supports hybrid cloud and edge computing. A flowchart details how the subsystems function within the IoT application.

- [View article](#)

Azure IoT reference architecture

- 9/10/2020
- 12 min read

Recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components

- [View article](#)

Process real-time vehicle data using IoT

- 11/17/2020
- 5 min read

This example builds a real-time data ingestion/processing pipeline to ingest and process messages from IoT devices into a big data analytic platform in Azure.

[View all](#)

Management and governance

AWS SERVICE	AZURE SERVICE	DESCRIPTION
AWS Organizations	Management Groups	Azure management groups help you organize your resources and subscriptions.
AWS Well-Architected Tool	Azure Well-Architected Review	Examine your workload through the lenses of reliability, cost management, operational excellence, security, and performance efficiency.
Trusted Advisor	Advisor	Provides analysis of cloud resource configuration and security, so that subscribers can ensure they're making use of best practices and optimum configurations.
AWS Billing and Cost Management	Azure Cost Management and Billing	Azure Cost Management and Billing helps you understand your Azure invoice (bill), manage your billing account and subscriptions, monitor and control Azure spending, and optimize resource use.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Cost and Usage Reports	Usage Details API	Services to help generate, monitor, forecast, and share billing data for resource usage by time, organization, or product resources.
Management Console	Portal	A unified management console that simplifies building, deploying, and operating your cloud resources.
Application Discovery Service	Migrate	Assesses on-premises workloads for migration to Azure, performs performance-based sizing, and provides cost estimations.
Systems Manager	Monitor	Comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.
Personal Health Dashboard	Resource Health	Provides detailed information about the health of resources, as well as recommended actions for maintaining resource health.
CloudTrail	Activity log	The Activity log is a platform log in Azure that provides insight into subscription-level events, such as when a resource is modified or when a virtual machine is started.
CloudWatch	Application Insights	A feature of Azure Monitor, Application Insights is an extensible Application Performance Management (APM) service for developers and DevOps professionals, which provides telemetry insights and information, in order to better understand how applications are performing and to identify areas for optimization.
Config	Application Change Analysis	Application Change Analysis detects various types of changes, from the infrastructure layer all the way to application deployment.
Cost Explorer	Cost Management	Optimize costs while maximizing cloud potential.
Control Tower	Blueprints	Set up and govern a multi account/subscription environment by creating landing zones.
Resource Groups and Tag Editor	Resource Groups and Tags	A Resource Group is a container that holds related resources for an Azure solution. Apply tags to your Azure resources to logically organize them by categories.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
AWS AppConfig	Azure App Configuration	Azure App Configuration is a managed service that helps developers centralize their application and feature settings simply and securely.
Service Catalog	Azure Managed Applications	Offers cloud solutions that are easy for consumers to deploy and operate.
SDKs and tools	SDKs and tools	Manage and interact with Azure services the way you prefer, programmatically from your language of choice, by using the Azure SDKs, our collection of tools, or both.

Messaging and eventing

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Simple Queue Service (SQS)	Queue Storage	Provides a managed message queueing service for communicating between decoupled application components.
Simple Notification Service (SNS)	Service Bus	Supports a set of cloud-based, message-oriented middleware technologies, including reliable message queuing and durable publish/subscribe messaging.
Amazon EventBridge	Event Grid	A fully managed event routing service that allows for uniform event consumption using a publish/subscribe model.
Amazon Kinesis	Event Hubs	A fully managed, real-time data ingestion service. Stream millions of events per second, from any source, to build dynamic data pipelines and to immediately respond to business challenges.
Amazon MQ	Service Bus	Service Bus Premium is fully compliant with the Java/Jakarta EE Java Message Service (JMS) 2.0 API. Service Bus Standard supports the JMS 1.1 subset focused on queues.

Messaging architectures

-

Anomaly Detector Process

- 12/16/2019
- 1 min read

Learn more about Anomaly Detector with a step-by-step flowchart that details the process. See how anomaly detection models are selected with time-series data.

- [REDACTED]

[Scalable web application](#)

- 10/03/2019
- 7 min read

Use the proven practices in this reference architecture to improve scalability and performance in an Azure App Service web application..

- [REDACTED]

[Enterprise integration using queues and events](#)

- 12/03/2018
- 5 min read

Recommended architecture for implementing an enterprise integration pattern with Azure Logic Apps, Azure API Management, Azure Service Bus, and Azure Event Grid.

- [REDACTED]

[Ops automation using Event Grid](#)

- 12/16/2019
- 1 min read

Event Grid allows you to speed automation and simplify policy enforcement. For example, Event Grid can notify Azure Automation when a virtual machine is created, or a SQL Database is spun up. These events can be used to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, or file work items.

Mobile services

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Mobile Hub	App Center Xamarin Apps	Provides backend mobile services for rapid development of mobile solutions, identity management, data synchronization, and storage and notifications across devices.
Mobile SDK	App Center	Provides the technology to rapidly build cross-platform and native apps for mobile devices.
Cognito	Azure Active Directory	Provides authentication capabilities for mobile applications.
Device Farm	App Center	Provides services to support testing mobile applications.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Mobile Analytics	App Center	Supports monitoring, and feedback collection for the debugging and analysis of a mobile application service quality.

Device Farm

The AWS Device Farm provides cross-device testing services. In Azure, [Visual Studio App Center](#) provides similar cross-device front-end testing for mobile devices.

In addition to front-end testing, the [Azure DevTest Labs](#) provides back-end testing resources for Linux and Windows environments.

Mobile architectures

- [Scalable web and mobile applications using Azure Database for PostgreSQL](#)

- 12/16/2019
- 1 min read

Use Azure Database for PostgreSQL to rapidly build engaging, performant, and scalable cross-platform and native apps for iOS, Android, Windows, or Mac.

- [Social App for Mobile and Web with Authentication](#)

- 12/16/2019
- 3 min read

View a detailed, step-by-step diagram depicting the build process and implementation of the mobile client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.

- [Task-Based Consumer Mobile App](#)

- 12/16/2019
- 3 min read

Learn how the task-based consumer mobile app architecture is created with a step-by-step flow chart that shows the integration with Azure App Service Mobile Apps, Visual Studio, and Xamarin to simplify the build process.

[view all](#)

Networking

AREA	AWS SERVICE	AZURE SERVICE	DESCRIPTION
------	-------------	---------------	-------------

Area	AWS Service	Azure Service	Description
Cloud virtual networking	Virtual Private Cloud (VPC)	Virtual Network	Provides an isolated, private environment in the cloud. Users have control over their virtual networking environment, including selection of their own IP address range, creation of subnets, and configuration of route tables and network gateways.
NAT gateways	NAT Gateways	Virtual Network NAT	A service that simplifies outbound-only Internet connectivity for virtual networks. When configured on a subnet, all outbound connectivity uses your specified static public IP addresses. Outbound connectivity is possible without a load balancer or public IP addresses directly attached to virtual machines.
Cross-premises connectivity	VPN Gateway	VPN Gateway	Connects Azure virtual networks to other Azure virtual networks, or customer on-premises networks (Site To Site). Allows end users to connect to Azure services through VPN tunneling (Point To Site).
DNS management	Route 53	DNS	Manage your DNS records using the same credentials and billing and support contract as your other Azure services
DNS-based routing	Route 53	Traffic Manager	A service that hosts domain names, plus routes users to Internet applications, connects user requests to datacenters, manages traffic to apps, and improves app availability with automatic failover.
Dedicated network	Direct Connect	ExpressRoute	Establishes a dedicated, private network connection from a location to the cloud provider (not over the Internet).

Area	AWS Service	Azure Service	Description
Load balancing	Network Load Balancer	Load Balancer	Azure Load Balancer load balances traffic at layer 4 (TCP or UDP). Standard Load Balancer also supports cross-region or global load balancing.
Application-level load balancing	Application Load Balancer	Application Gateway	Application Gateway is a layer 7 load balancer. It supports SSL termination, cookie-based session affinity, and round robin for load-balancing traffic.
Route table	Custom Route Tables	User Defined Routes	Custom, or user-defined (static) routes to override default system routes, or to add more routes to a subnet's route table.
Private link	PrivateLink	Azure Private Link	Azure Private Link provides private access to services that are hosted on the Azure platform. This keeps your data on the Microsoft network.
Private PaaS connectivity	VPC endpoints	Private Endpoint	Private Endpoint provides secured, private connectivity to various Azure platform as a service (PaaS) resources, over a backbone Microsoft private network.
Virtual network peering	VPC Peering	VNET Peering	VNet peering is a mechanism that connects two virtual networks (VNets) in the same region through the Azure backbone network. Once peered, the two virtual networks appear as one for all connectivity purposes.
Content delivery networks	Cloud Front	Azure CDN	The Azure Content Delivery Network is designed to send audio, video, apps, photos, and other files to your customers faster and more reliably, using the servers closest to each user. Acceleration Data Transfer provides dynamic site acceleration of non-cacheable, dynamic content that is generated by your web applications.

Area	AWS Service	Azure Service	Description
Network Monitoring	VPC Flow Logs	Azure Network Watcher	Azure Network Watcher allows you to monitor, diagnose, and analyze the traffic in Azure Virtual Network.

Networking architectures

- [Deploy highly available NVAs](#)

Deploy highly available NVAs

- 12/08/2018
- 7 min read

Learn how to deploy network virtual appliances for high availability in Azure. This article includes example architectures for ingress, egress, and both.

- [Hub-spoke network topology in Azure](#)

- 9/30/2020
- 7 min read

Learn how to implement a hub-spoke topology in Azure, where the hub is a virtual network and the spokes are virtual networks that peer with the hub.

- [Implement a secure hybrid network](#)

- 1/07/2020
- 9 min read

See a secure hybrid network that extends an on-premises network to Azure with a perimeter network between the on-premises network and an Azure virtual network.

[View all](#)

Security, identity, and access

Authentication and authorization

AWS Service	Azure Service	Description
Identity and Access Management (IAM)	Azure Active Directory	Allows users to securely control access to services and resources while offering data security and protection. Create and manage users and groups, and use permissions to allow and deny access to resources.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Identity and Access Management (IAM)	Azure role-based access control	Azure role-based access control (Azure RBAC) helps you manage who has access to Azure resources, what they can do with those resources, and what areas they have access to.
Organizations	Subscription Management + Azure RBAC	Security policy and role management for working with multiple accounts.
Multi-Factor Authentication	Azure Active Directory	Safeguard access to data and applications, while meeting user demand for a simple sign-in process.
Directory Service	Azure Active Directory Domain Services	Provides managed domain services, such as domain join, group policy, LDAP, and Kerberos/NTLM authentication, which are fully compatible with Windows Server Active Directory.
Cognito	Azure Active Directory B2C	A highly available, global, identity management service for consumer-facing applications that scales to hundreds of millions of identities.
Organizations	Policy	Azure Policy is a service in Azure that you use to create, assign, and manage policies. These policies enforce different rules and effects over your resources, so those resources stay compliant with your corporate standards and service level agreements.
Organizations	Management Groups	Azure management groups provide a level of scope above subscriptions. You organize subscriptions into containers called "management groups" and apply your governance conditions to the management groups. All subscriptions within a management group automatically inherit the conditions applied to the management group. Management groups give you enterprise-grade management at a large scale, no matter what type of subscriptions you have.

Encryption

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Server-side encryption with Amazon S3 Key Management Service	Azure Storage Service Encryption	Helps you protect and safeguard your data and meet your organizational security and compliance commitments.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Key Management Service (KMS), CloudHSM	Key Vault	Provides security solution and works with other services by providing a way to manage, create, and control encryption keys stored in hardware security modules (HSM).

Firewall

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Web Application Firewall	Web Application Firewall	A firewall that protects web applications from common web exploits.
Web Application Firewall	Firewall	Provides inbound protection for non-HTTP/S protocols, outbound network-level protection for all ports and protocols, and application-level protection for outbound HTTP/S.

Security

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Inspector	Defender for Cloud	An automated security assessment service that improves the security and compliance of applications. Automatically assess applications for vulnerabilities or deviations from best practices.
Certificate Manager	App Service Certificates available on the Portal	Service that allows customers to create, manage, and consume certificates seamlessly in the cloud.
GuardDuty	Advanced Threat Protection	Detect and investigate advanced attacks on-premises and in the cloud.
Artifact	Service Trust Portal	Provides access to audit reports, compliance guides, and trust documents from across cloud services.
Shield	DDoS Protection Service	Provides cloud services with protection from distributed denial of services (DDoS) attacks.

Security architectures

-

Real-time fraud detection

- 7/05/2018
- 4 min read

Detect fraudulent activity in real-time using Azure Event Hubs and Stream Analytics.



Securely managed web applications

- 5/09/2019
- 8 min read

Learn about deploying secure applications using the Azure App Service Environment, the Azure Application Gateway service, and Web Application Firewall.



Threat indicators for cyber threat intelligence in Azure Sentinel

- 4/13/2020
- 13 min read

Import threat indicators, view logs, create rules to generate security alerts and incidents, and visualize threat intelligence data with Azure Sentinel.

[view all](#)

Storage

Object storage

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Simple Storage Services (S3)	Blob storage	Object storage service, for use cases including cloud applications, content distribution, backup, archiving, disaster recovery, and big data analytics.

Virtual server disks

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Block Store (EBS)	managed disks	SSD storage optimized for I/O intensive read/write operations. For use as high-performance Azure virtual machine storage.

Shared files

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic File System	Files	Provides a simple interface to create and configure file systems quickly, and share common files. Can be used with traditional protocols that access files over a network.

Archiving and backup

AWS SERVICE	AZURE SERVICE	DESCRIPTION
S3 Infrequent Access (IA)	Storage cool tier	Cool storage is a lower-cost tier for storing data that is infrequently accessed and long-lived.
S3 Glacier, Deep Archive	Storage archive access tier	Archive storage has the lowest storage cost and higher data retrieval costs compared to hot and cool storage.
Backup	Backup	Back up and recover files and folders from the cloud, and provide offsite protection against data loss.

Hybrid storage

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Storage Gateway	StorSimple	Integrates on-premises IT environments with cloud storage. Automates data management and storage, plus supports disaster recovery.
DataSync	File Sync	Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync.

Bulk data transfer

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Import/Export Disk	Import/Export	A data transport solution that uses secure disks and appliances to transfer large amounts of data. Also offers data protection during transit.
Import/Export Snowball, Snowball Edge, Snowmobile	Data Box	Petabyte- to exabyte-scale data transport solution that uses secure data storage devices to transfer large amounts of data to and from Azure.

Storage architectures

-

HIPAA and HITRUST compliant health data AI

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.

-

HPC Media Rendering

- 11/04/2020
- 2 min read

Optimize the media rendering process with a step-by-step HPC solution architecture from Azure that combines Azure CycleCloud and HPC Cache.

- [View article](#)

Medical Data Storage Solutions

- 12/16/2019
- 2 min read

Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security.

[view all](#)

Web applications

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Elastic Beanstalk	App Service	Managed hosting platform providing easy to use services for deploying and scaling web applications and services.
API Gateway	API Management	A turnkey solution for publishing APIs to external and internal consumers.
CloudFront	Content Delivery Network	A global content delivery network that delivers audio, video, applications, images, and other files.
Global Accelerator	Front Door	Easily join your distributed microservices architectures into a single global application using HTTP load balancing and path-based routing rules. Automate turning up new regions and scale-out with API-driven global actions, and independent fault-tolerance to your back end microservices in Azure-or anywhere.
Global Accelerator	Cross-regional load balancer	Distribute and load balance traffic across multiple Azure regions via a single, static, global anycast public IP address.
LightSail	App Service	Build, deploy, and scale web apps on a fully managed platform.
App Runner	Web App for Containers	Easily deploy and run containerized web apps on Windows and Linux.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
Amplify	Static Web Apps	Boost productivity with a tailored developer experience, CI/CD workflows to build and deploy your static content hosting, and dynamic scale for integrated serverless APIs.

Web architectures

- [Architect scalable e-commerce web app](#)

[Architect scalable e-commerce web app](#)

- 12/16/2019
- 1 min read

The e-commerce website includes simple order processing workflows with the help of Azure services. Using Azure Functions and Web Apps, developers can focus on building personalized experiences and let Azure take care of the infrastructure.

- [Multi-region N-tier application](#)

[Multi-region N-tier application](#)

- 6/18/2019
- 10 min read

Deploy an application on Azure virtual machines in multiple regions for high availability and resiliency.

- [Serverless web application](#)

- 5/28/2019
- 16 min read

This reference architecture shows a serverless web application, which serves static content from Azure Blob Storage and implements an API using Azure Functions.

[view all](#)

End-user computing

AWS SERVICE	AZURE SERVICE	DESCRIPTION
WorkSpaces, AppStream 2.0	Azure Virtual Desktop	Manage virtual desktops and applications to enable corporate network and data access to users, anytime, anywhere, from supported devices. Amazon WorkSpaces support Windows and Linux virtual desktops. Azure Virtual Desktop supports multi-session Windows 10 virtual desktops.

AWS SERVICE	AZURE SERVICE	DESCRIPTION
WorkLink	Application Proxy	Provides access to intranet applications, without requiring VPN connectivity. Amazon WorkLink is limited to iOS and Android devices.

Miscellaneous

AREA	AWS SERVICE	AZURE SERVICE	DESCRIPTION
Backend process logic	Step Functions	Logic Apps	Cloud technology to build distributed applications using out-of-the-box connectors to reduce integration challenges. Connect apps, data, and devices on-premises or in the cloud.
Enterprise application services	WorkMail , WorkDocs , Chime	Microsoft 365	Fully integrated cloud service that provides communications, email, and document management in the cloud and is available on a wide variety of devices.
Gaming	GameLift	PlayFab	Managed services for hosting dedicated game servers.
Media transcoding	Elastic Transcoder	Media Services	Services that offer broadcast-quality video streaming services, including various transcoding technologies.
Workflow	Step Functions	Logic Apps	Serverless technology for connecting apps, data and devices anywhere, whether on-premises or in the cloud for large ecosystems of SaaS and cloud-based connectors.
Hybrid	Outposts	Stack	Azure Stack is a hybrid cloud platform that enables you to run Azure services in your company's or service provider's datacenter. As a developer, you can build apps on Azure Stack. You can then deploy them to either Azure Stack or Azure, or you can build truly hybrid apps that take advantage of connectivity between an Azure Stack cloud and Azure.

AREA	AWS SERVICE	AZURE SERVICE	DESCRIPTION
Media	Elemental MediaConvert	Media Services	Cloud-based media workflow platform to index, package, protect, and stream video at scale.
Satellite	Ground Station	Azure Orbital	Fully managed cloud-based ground station as a service.
Quantum computing	Amazon Braket	Azure Quantum	Managed quantum computing service that developers, researchers, and businesses can use to run quantum computing programs.

More learning

If you are new to Azure, review the interactive [Core Cloud Services - Introduction to Azure](#) module on [Microsoft Learn](#).

Azure for Google Cloud Professionals

3/10/2022 • 9 minutes to read • [Edit Online](#)

This article helps Google Cloud experts understand the basics of Microsoft Azure accounts, platform, and services. It also covers key similarities and differences between the Google Cloud and Azure platforms. (Note that Google Cloud was previously called Google Cloud Platform (GCP).)

You'll learn:

- How accounts and resources are organized in Azure.
- How available solutions are structured in Azure.
- How the major Azure services differ from Google Cloud services.

Azure and Google Cloud built their capabilities independently over time so that each has important implementation and design differences.

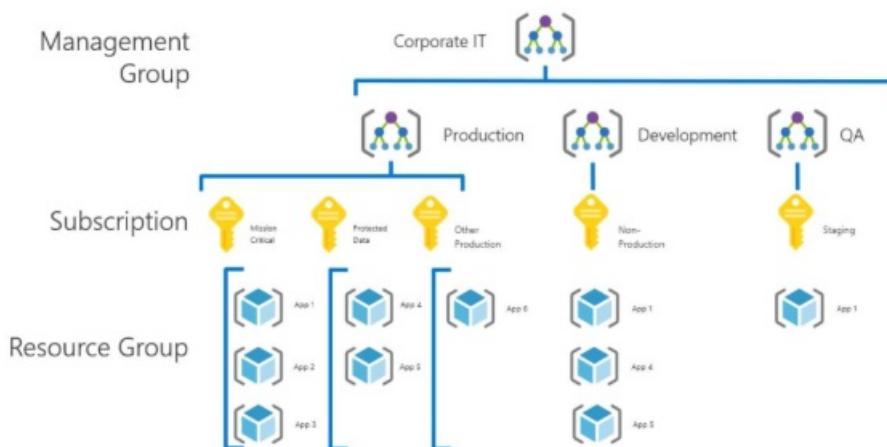
Azure for Google Cloud overview

Like Google Cloud, Microsoft Azure is built around a core set of compute, storage, database, and networking services. In many cases, both platforms offer a basic equivalence between the products and services they offer. Both Google Cloud and Azure allow you to build highly available solutions based on Linux or Windows hosts. So, if you're used to development using Linux and OSS technology, both platforms can do the job.

While the capabilities of both platforms are similar, the resources that provide those capabilities are often organized differently. Exact one-to-one relationships between the services required to build a solution are not always clear. In other cases, a particular service might be offered on one platform, but not the other.

Managing accounts and subscription

Azure has a hierarchy of Management group and subscriptions and resource groups to manage resources effectively. This is similar to the Folders and Project hierarchy for resources in Google Cloud.



Azure levels of management scope

- **Management groups:** These groups are containers that help you manage access, policy, and compliance for multiple subscriptions. All subscriptions in a management group automatically inherit the conditions applied to the management group.
- **Subscriptions:** A subscription logically associates user accounts and the resources that were created by

those user accounts. Each subscription has limits or quotas on the amount of resources you can create and use. Organizations can use subscriptions to manage costs and the resources that are created by users, teams, or projects.

- **Resource groups:** A resource group is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.
- **Resources:** Resources are instances of services that you create, like virtual machines, storage, or SQL databases.

Azure services can be purchased using several pricing options, depending on your organization's size and needs. See the [pricing overview](#) page for details.

[Azure subscriptions](#) are a grouping of resources with an assigned owner responsible for billing and permissions management.

A Google Cloud *project* is conceptually similar to the Azure subscription, in terms of billing, quotas, and limits. However, from a functional perspective, a Google Cloud project is more like a resource group in Azure. It's a logical unit that cloud resources are deployed to.

Note that unlike in Google Cloud, there is no maximum number of Azure subscriptions. Each Azure subscription is linked to a single Azure Active Directory (Azure AD) tenant (an *account*, in Google Cloud terms). An Azure AD tenant can contain an unlimited number of subscriptions, whereas Google Cloud has a default limit of 30 projects per account.

Subscriptions are assigned three types of administrator accounts:

- **Account Administrator.** The subscription owner and the account billed for the resources used in the subscription. The account administrator can only be changed by transferring ownership of the subscription.
- **Service Administrator.** This account has rights to create and manage resources in the subscription but is not responsible for billing. By default, the account administrator and service administrator are assigned to the same account. The account administrator can assign a separate user to the service administrator account for managing the technical and operational aspects of a subscription. Only one service administrator is assigned per subscription.
- **Co-administrator.** There can be multiple co-administrator accounts assigned to a subscription. Co-administrators cannot change the service administrator, but otherwise have full control over subscription resources and users.

For fine-grained access management to Azure resources, you can use Azure role-based access control ([Azure RBAC](#)), which includes over 70 built-in roles. You can also create your own custom roles.

Below the subscription level user roles and individual permissions can also be assigned to specific resources. In Azure, all user accounts are associated with either a Microsoft Account or Organizational Account (an account managed through Azure AD).

Subscriptions have default service quotas and limits. For a full list of these limits, see [Azure subscription and service limits, quotas, and constraints](#). These limits can be increased up to the maximum by [filing a support request in the management portal](#).

See also

- [How to add or change Azure administrator roles](#)
- [How to download your Azure billing invoice and daily usage data](#)

Resource management

The term "resource" in Azure means any compute instance, storage object, networking device, or other entity you can create or configure within the platform.

Azure resources are deployed and managed using one of two models: [Azure Resource Manager](#), or the older Azure [classic deployment model](#). Any new resources are created using the Resource Manager model.

Resource groups

Azure additionally has an entity called "resource groups" that organize resources such as VMs, storage, and virtual networking devices. An Azure resource is always associated with one resource group. A resource created in one resource group can be moved to another group but can only be in one resource group at a time. For more information, see [Move Azure resources across resource groups, subscriptions, or regions](#). Resource groups are the fundamental grouping used by Azure Resource Manager.

Resources can also be organized using [tags](#). Tags are key-value pairs that allow you to group resources across your subscription irrespective of resource group membership.

Management interfaces

Azure offers several ways to manage your resources:

- [Web interface](#). The Azure portal provides a full web-based management interface for Azure resources.
- [REST API](#). The Azure Resource Manager REST API provides programmatic access to most of the features available in the Azure portal.
- [Command Line](#). The Azure CLI provides a command-line interface capable of creating and managing Azure resources. The Azure CLI is available for [Windows, Linux, and macOS](#).
- [PowerShell](#). The Azure modules for PowerShell allow you to execute automated management tasks using a script. PowerShell is available for [Windows, Linux, and macOS](#).
- [Templates](#). Azure Resource Manager templates provide JSON template-based resource management capabilities.
- [SDK](#). The SDKs are a collection of libraries that allows users to programmatically manage and interact with Azure services.

In each of these interfaces, the resource group is central to how Azure resources get created, deployed, or modified.

In addition, many third-party management tools like [Hashicorp's Terraform](#) and [Netflix Spinnaker](#), are also available on Azure.

See also

- [Azure resource group guidelines](#)

Regions and Availability Zones

Failures can vary in the scope of their impact. Some hardware failures, such as a failed disk, may affect a single host machine. A failed network switch could affect a whole server rack. Less common are failures that disrupt a whole datacenter, such as loss of power in a datacenter. In rare situations, an entire region could become unavailable.

One of the main ways to make an application resilient is through redundancy. However, you need to plan for this redundancy when you design the application. Also, the level of redundancy that you need depends on your business requirements. Not every application needs redundancy across regions to guard against a regional outage. In general, a tradeoff exists between greater redundancy and reliability versus higher cost and complexity.

In Google Cloud, a region has two or more Availability Zones. An Availability Zone corresponds with a physically isolated datacenter in the geographic region. Azure has numerous features for providing application redundancy at every level of potential failure, including [availability sets](#), [availability zones](#), and [paired regions](#).

The following table summarizes each option.

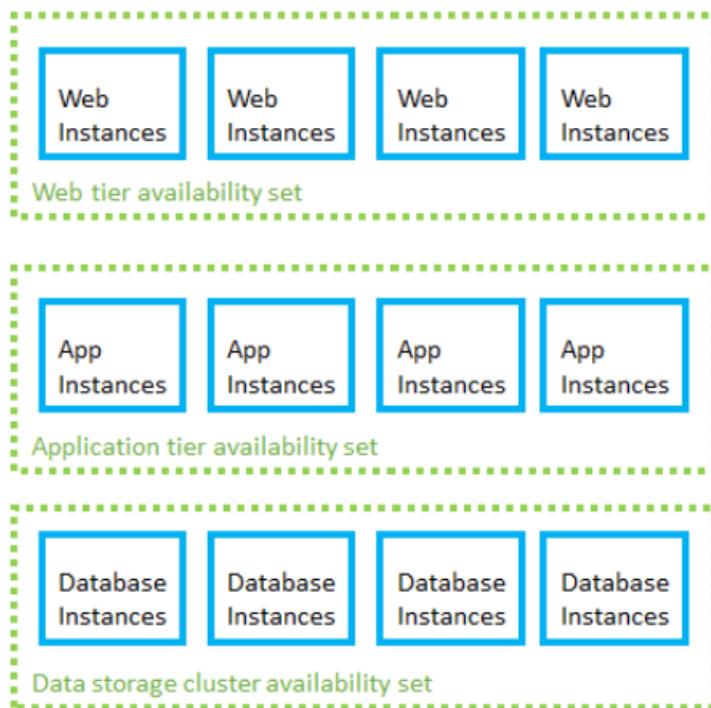
	AVAILABILITY SET	AVAILABILITY ZONE	PAIRED REGION
Scope of failure	Rack	Datacenter	Region
Request routing	Load Balancer	Cross-zone Load Balancer	Traffic Manager
Network latency	Very low	Low	Mid to high
Virtual networking	VNet	VNet	Cross-region VNet peering

Availability sets

To protect against localized hardware failures, such as a disk or network switch failing, deploy two or more VMs in an availability set. An availability set consists of two or more *fault domains* that share a common power source and network switch. VMs in an availability set are distributed across the fault domains, so if a hardware failure affects one fault domain, network traffic can still be routed to the VMs in the other fault domains. For more information about Availability Sets, see [Manage the availability of Windows virtual machines in Azure](#).

When VM instances are added to availability sets, they are also assigned an [update domain](#). An update domain is a group of VMs that are set for planned maintenance events at the same time. Distributing VMs across multiple update domains ensures that planned update and patching events affect only a subset of these VMs at any given time.

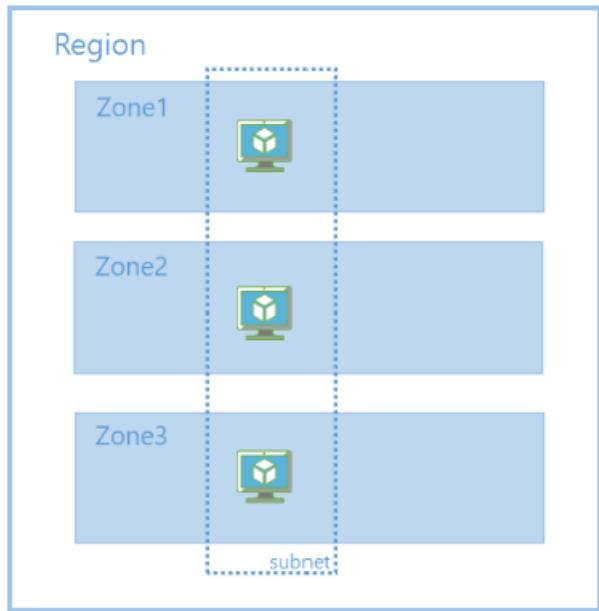
Availability sets should be organized by the instance's role in your application to ensure one instance in each role is operational. For example, in a three-tier web application, create separate availability sets for the front-end, application, and data tiers.



Availability sets

Availability Zones

Like Google Cloud, Azure regions can have Availability zones. An [Availability Zone](#) is a physically separate zone within an Azure region. Each Availability Zone has a distinct power source, network, and cooling. Deploying VMs across availability zones helps to protect an application against datacenter-wide failures.



Zone redundant VM deployment on Azure

For more information, see [Build solutions for high availability using Availability Zones](#).

Paired regions

To protect an application against a regional outage, you can deploy the application across multiple regions, using [Azure Traffic Manager](#) to distribute internet traffic to the different regions. Each Azure region is paired with another region. Together, these form a [regional pair](#). With the exception of Brazil South, regional pairs are located within the same geography in order to meet data residency requirements for tax and law enforcement jurisdiction purposes.

Unlike Availability Zones, which are physically separate datacenters but may be in relatively nearby geographic areas, paired regions are typically separated by at least 300 miles. This design ensures that large-scale disasters only affect one of the regions in the pair. Neighboring pairs can be set to sync database and storage service data, and are configured so that platform updates are rolled out to only one region in the pair at a time.

Azure [geo-redundant storage](#) is automatically backed up to the appropriate paired region. For all other resources, creating a fully redundant solution using paired regions means creating a full copy of your solution in both regions.



Region Pairs in Azure

See also

- [Regions for virtual machines in Azure](#)
- [Availability options for virtual machines in Azure](#)
- [High availability for Azure applications](#)
- [Failure and disaster recovery for Azure applications](#)

- Planned maintenance for Linux virtual machines in Azure

Services

For a listing of how services map between platforms, see [Google Cloud to Azure services comparison](#).

Not all Azure products and services are available in all regions. Consult the [Products by Region](#) page for details. You can find the uptime guarantees and downtime credit policies for each Azure product or service on the [Service Level Agreements](#) page.

Next steps

- [Get started with Azure](#)
- [Azure solution architectures](#)
- [Azure Reference Architectures](#)

Google Cloud to Azure services comparison

3/10/2022 • 31 minutes to read • [Edit Online](#)

This article helps you understand how Microsoft Azure services compare to Google Cloud. (Note that Google Cloud used to be called the Google Cloud Platform (GCP).) Whether you are planning a multi-cloud solution with Azure and Google Cloud, or migrating to Azure, you can compare the IT capabilities of Azure and Google Cloud services in all the technology categories.

This article compares services that are roughly comparable. Not every Google Cloud service or Azure service is listed, and not every matched service has exact feature-for-feature parity.

For an overview of Azure for Google Cloud users, see the introduction to [Azure for Google Cloud Professionals](#).

Marketplace

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Google Cloud Marketplace	Azure Marketplace	Easy-to-deploy and automatically configured third-party applications, including single virtual machine or multiple virtual machine solutions.

Data platform

Database

TYPE	GOOGLE CLOUD SERVICE	AZURE SERVICE	AZURE SERVICE DESCRIPTION

Type	Google Cloud Service	Azure Service	Azure Service Description
Relational database	Cloud SQL - SQL Server	Azure SQL Server Family Azure SQL Database Azure SQL Managed Instance SQL Server on Azure VM Azure SQL Edge	Azure SQL family of SQL Server database engine products in the cloud Azure SQL Database is a fully managed platform as a service (PaaS) database engine Azure SQL Managed Instance is the intelligent, scalable cloud database service that combines the broadest SQL Server database engine compatibility with all the benefits of a fully managed and evergreen platform as a service SQL Server IaaS deployed on Azure Windows or Linux VM Azure SQL Edge is an optimized relational database engine geared for IoT and edge deployments
	Cloud SQL MySQL & PostgreSQL	Azure Database for MySQL (Single & Flexible Server) Azure Database for PostgreSQL (Single & Flexible Server)	Managed relational database service where resiliency, security, scale, and maintenance are primarily handled by the platform
Horizontally scalable relational database	Cloud Spanner	Azure Cosmos DB - SQL API	A globally-distributed database system that limitlessly scales horizontally. Is multi-modal -- key-value, graph, and document data). Supports multiple APIs: SQL, JavaScript, Gremlin, MongoDB, and Azure Table storage. Compute and storage can be scaled independently

Type	Google Cloud Service	Azure Service	Azure Service Description
		Azure PostgreSQL Hyperscale (Citus)	Azure Database for PostgreSQL is a fully managed database-as-a-service based on the open-source Postgres relational database engine. The Hyperscale (Citus) deployment option scales queries across multiple machines using sharding, to serve applications that require greater scale and performance
NoSQL	Cloud Bigtable	Azure Table storage	A highly scalable NoSQL key-value store for rapid development using massive semi-structured datasets. Store semi-structured data that's highly available. Supporting flexible data schema and OData-based queries
	Cloud Firestore	Azure Cosmos DB	Globally distributed, multi-model database that natively supports multiple data models: key-value, documents, graphs, and columnar
	Firebase Realtime Database	Azure Cosmos DB - Change Feed	Change feed in Azure Cosmos DB is a persistent record of changes to a container in the order they occur. Change feed support in Azure Cosmos DB works by listening to an Azure Cosmos container for any changes. It then outputs the sorted list of documents that were changed in the order in which they were modified. The persisted changes can be processed asynchronously and incrementally, and the output can be distributed across one or more consumers for parallel processing
In-memory	Cloud Memorystore	Azure Cache for Redis	A secure data cache and messaging broker that provides high throughput and low-latency access to data for applications

TYPE	GOOGLE CLOUD SERVICE	AZURE SERVICE	AZURE SERVICE DESCRIPTION

Database architectures

-

Gaming using Cosmos DB

- 12/16/2019
- 1 min read

Elastically scale your database to accommodate unpredictable bursts of traffic and deliver low-latency multi-player experiences on a global scale.

-

Oracle Database Migration to Azure

- 12/16/2019
- 2 min read

Oracle DB migrations can be accomplished in multiple ways. This architecture covers one of these options wherein Oracle Active Data Guard is used to migrate the Database.

-

Retail and e-commerce using Azure MySQL

- 12/16/2019
- 1 min read

Build secure and scalable e-commerce solutions that meet the demands of both customers and business using Azure Database for MySQL.

[view all](#)

Data warehouse

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
BigQuery	Azure Synapse Analytics SQL Server Big Data Clusters Azure Databricks	<p>Cloud-based Enterprise Data Warehouse (EDW) that uses Massively Parallel Processing (MPP) to quickly run complex queries across petabytes of data.</p> <p>Allow you to deploy scalable clusters of SQL Server, Spark, and HDFS containers running on Kubernetes. These components are running side by side to enable you to read, write, and process big data from Transact-SQL or Spark, allowing you to easily combine and analyze your high-value relational data with high-volume big data.</p>

Data warehouse architectures

- [View all](#)

Modern Data Warehouse Architecture

- 12/16/2019
- 2 min read

Explore a cloud data warehouse that uses big data. Modern data warehouse brings together all your data and scales easily as your data grows.

- [View all](#)

Automated enterprise BI

- 6/03/2020
- 13 min read

Automate an extract, load, and transform (ELT) workflow in Azure using Azure Data Factory with Azure Synapse Analytics.

[View all](#)

Data orchestration and ETL

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Data Fusion	Azure Data Factory Azure Synapse Analytics	Processes and moves data between different compute and storage services, as well as on-premises data sources at specified intervals. Create, schedule, orchestrate, and manage data pipelines.

Big data and analytics

Big data processing

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Dataproc	Azure HDInsight Azure Synapse Analytics Azure Databricks	Managed Apache Spark-based analytics platform.

Big data architectures

- [View all](#)

Azure data platform end-to-end

- 1/31/2020
- 7 min read

Use Azure services to ingest, process, store, serve, and visualize data from different sources.

- [View all](#)

Campaign Optimization with Azure HDInsight Spark Clusters

- 12/16/2019
- 4 min read

This solution demonstrates how to build and deploy a machine learning model with Microsoft R Server on Azure HDInsight Spark clusters to recommend actions to maximize the purchase rate of leads targeted by a campaign. This solution enables efficient handling of big data on Spark with Microsoft R Server.

- [REDACTED]

Big data analytics with Azure Data Explorer

- 8/11/2020
- 3 min read

Big Data Analytics with Azure Data Explorer demonstrates Azure Data Explorer's abilities to cater to volume, velocity, and variety of data, the three V's of big data.

[view all](#)

Analytics and visualization

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Dataflow	Azure Databricks	Managed platform for streaming batch data based on Open Source Apache products.
Data Studio Looker	Power BI	Business intelligence tools that build visualizations, perform ad hoc analysis, and develop business insights from data.
Cloud Search	Azure Search	Delivers full-text search and related search analytics and capabilities.
BigQuery	SQL Server Analysis Services	Provides a serverless non-cloud interactive query service that uses standard SQL for analyzing databases.

Analytics architectures

- [REDACTED]

Advanced Analytics Architecture

- 12/16/2019
- 2 min read

Get near real-time data analytics on streaming services. This big data architecture allows you to combine any data at any scale with custom machine learning.

- [REDACTED]

Automated enterprise BI

- 6/03/2020

- 13 min read

Automate an extract, load, and transform (ELT) workflow in Azure using Azure Data Factory with Azure Synapse Analytics.

- [REDACTED]

[Mass ingestion and analysis of news feeds on Azure](#)

- 2/01/2019
- 5 min read

Create a pipeline for ingesting and analyzing text, images, sentiment, and other data from RSS news feeds using only Azure services, including Azure Cosmos DB and Azure Cognitive Services.

[view all](#)

Time series & IoT data

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
BigQuery	Azure Data Explorer Azure Time Series Insights Cosmos DB	<p>Fully managed, low latency, and distributed big data analytics platform that runs complex queries across petabytes of data. Highly optimized for log and time series data.</p> <p>Open and scalable end-to-end IoT analytics service. Collect, process, store, query, and visualize data at Internet of Things (IoT) scale--data that's highly contextualized and optimized for time series.</p>

Time series architectures

- [REDACTED]

[IoT analytics with Azure Data Explorer](#)

- 8/11/2020
- 3 min read

IoT Telemetry Analytics with Azure Data Explorer demonstrates near real-time analytics over fast flowing, high volume, wide variety of streaming data from IoT devices.

- [REDACTED]

[Azure Data Explorer interactive analytics](#)

- 8/11/2020
- 3 min read

Interactive Analytics with Azure Data Explorer focuses on its integration with the rest of the data platform ecosystem.

AI and machine learning

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Vertex AI	Azure Machine Learning	A cloud service to train, deploy, automate, and manage machine learning models.
TensorFlow	ML.NET	ML.NET is an open source and cross-platform machine learning framework for both machine learning & AI.
TensorFlow	ONNX (Open Neural Network Exchange)	ONNX is an open format built to represent machine learning models that facilitates maximum compatibility and increased inference performance.
Vision AI	Azure Cognitive Services Computer Vision	Use visual data processing to label content, from objects to concepts, extract printed and handwritten text, recognize familiar subjects like brands and landmarks, and moderate content. No machine learning expertise is required.
Natural Language AI	Azure Cognitive Services Text Analytics	Cloud-based services that provides advanced natural language processing over raw text, and includes four main functions: sentiment analysis, key phrase extraction, language detection, and named entity recognition.
Natural Language AI	Azure Cognitive Services Language Understanding (LUIS)	A machine learning-based service to build natural language understanding into apps, bots, and IoT devices. Quickly create enterprise-ready, custom models that continuously improve.
Speech-to-Text	Azure Cognitive Services Speech To Text	Swiftly convert audio into text from a variety of sources. Customize models to overcome common speech recognition barriers, such as unique vocabularies, speaking styles, or background noise.
AutoML Tables – Structured Data	Azure ML - Automated Machine Learning	Empower professional and non-professional data scientists to build machine learning models rapidly. Automate time-consuming and iterative tasks of model development using breakthrough research-and accelerate time to market. Available in Azure Machine learning, Power BI, ML.NET & Visual Studio.

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
AutoML Tables – Structured Data	ML.NET Model Builder	ML.NET Model Builder provides an easy to understand visual interface to build, train, and deploy custom machine learning models. Prior machine learning expertise is not required. Model Builder supports AutoML, which automatically explores different machine learning algorithms and settings to help you find the one that best suits your scenario.
AutoML Vision	Azure Cognitive Services Custom Vision	Customize and embed state-of-the-art computer vision for specific domains. Build frictionless customer experiences, optimize manufacturing processes, accelerate digital marketing campaigns-and more. No machine learning expertise is required.
AutoML Video Intelligence	Azure Video Analyzer	Easily extract insights from your videos and quickly enrich your applications to enhance discovery and engagement.
Dialogflow	Azure Cognitive Services QnA Maker	Build, train and publish a sophisticated bot using FAQ pages, support websites, product manuals, SharePoint documents or editorial content through an easy-to-use UI or via REST APIs.
AI Platform Notebooks	Azure Notebooks	Develop and run code from anywhere with Jupyter notebooks on Azure.
Deep Learning VM Image	Data Science Virtual Machines	Pre-Configured environments in the cloud for Data Science and AI Development.
Deep Learning Containers	GPU support on Azure Kubernetes Service (AKS)	Graphical processing units (GPUs) are often used for compute-intensive workloads such as graphics and visualization workloads. AKS supports the creation of GPU-enabled node pools to run these compute-intensive workloads in Kubernetes.
Data Labeling Service	Azure ML - Data Labeling	A central place to create, manage, and monitor labeling projects (public preview). Use it to coordinate data, labels, and team members to efficiently manage labeling tasks. Machine Learning supports image classification, either multi-label or multi-class, and object identification with bounded boxes.

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
AI Platform Training	Azure ML – Compute Targets	Designated compute resource/environment where you run your training script or host your service deployment. This location may be your local machine or a cloud-based compute resource. Using compute targets make it easy for you to later change your compute environment without having to change your code.
AI Platform Predictions	Azure ML - Deployments	Deploy your machine learning model as a web service in the Azure cloud or to Azure IoT Edge devices. Leverage serverless Azure Functions for model inference for dynamic scale.
Continuous Evaluation	Azure ML – Data Drift	Monitor for data drift between the training dataset and inference data of a deployed model. In the context of machine learning, trained machine learning models may experience degraded prediction performance because of drift. With Azure Machine Learning, you can monitor data drift and the service can send an email alert to you when drift is detected.
What-If Tool	Azure ML – Model Interpretability	Ensure machine learning model compliance with company policies, industry standards, and government regulations.
Cloud TPU	Azure ML – FPGA (Field Programmable Gate Arrays)	FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects. The interconnects allow these blocks to be configured in various ways after manufacturing. Compared to other chips, FPGAs provide a combination of programmability and performance.
Kubeflow	Machine Learning Operations (MLOps)	MLOps, or DevOps for machine learning, enables data science and IT teams to collaborate and increase the pace of model development and deployment via monitoring, validation, and governance of machine learning models.
Dialogflow	Microsoft Bot Framework	Build and connect intelligent bots that interact with your users using text/SMS, Skype, Teams, Slack, Microsoft 365 mail, Twitter, and other popular services.

AI and machine learning architectures



[Image classification on Azure](#)

- 7/05/2018
- 4 min read

Learn how to build image processing into your applications by using Azure services such as the Computer Vision API and Azure Functions.



[Predictive Marketing with Machine Learning](#)

- 12/16/2019
- 2 min read

Learn how to build a machine-learning model with Microsoft R Server on Azure HDInsight Spark clusters to recommend actions to maximize the purchase rate.



[Scalable personalization on Azure](#)

- 5/31/2019
- 6 min read

Use machine learning to automate content-based personalization for customers.

[view all](#)

Data catalog & governance

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Data Catalog	Azure Purview	Azure Purview is a unified data governance service that helps you manage and govern your on-premises, multi-cloud, and software-as-a-service (SaaS) data.

Compute

Virtual servers

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Compute Engine	Azure Virtual Machines	Virtual servers allow users to deploy, manage, and maintain OS and server software. Instance types provide combinations of CPU/RAM. Users pay for what they use with the flexibility to change sizes.
Sole-tenant nodes	Azure Dedicated Host	Host your VMs on hardware that's dedicated only to your project.

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Batch	Azure Batch	Run large-scale parallel and high-performance computing applications efficiently in the cloud.
Compute Engine Autoscaler	Azure virtual machine scale sets	Allows you to automatically change the number of VM instances. You set defined metric and thresholds that determine if the platform adds or removes instances.
Compute Engine managed instance groups		
Cloud GPUs	GPU Optimized VMs	GPU-optimized VM sizes are specialized virtual machines that are available with single, multiple, or fractional GPUs. The sizes are designed for compute-intensive, graphics-intensive, and visualization workloads.
VMware Engine	Azure VMware Solution	Redeploy and extend your VMware-based enterprise workloads to Azure with Azure VMware Solution. Seamlessly move VMware-based workloads from your datacenter to Azure and integrate your VMware environment with Azure. Keep managing your existing environments with the same VMware tools that you already know, while you modernize your applications with Azure native services. Azure VMware Solution is a Microsoft service that is verified by VMware, and it runs on Azure infrastructure.

Containers and container orchestrators

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Run	Azure Container Instances	Azure Container Instances is the fastest and simplest way to run a container in Azure, without having to provision any virtual machines or adopt a higher-level orchestration service.
Artifact Registry (beta)	Azure Container Registry	Allows customers to store Docker formatted images. Used to create all types of container deployments on Azure.
Container Registry		
Kubernetes Engine (GKE)	Azure Kubernetes Service (AKS)	Deploy orchestrated containerized applications with Kubernetes. Simplify cluster management and monitoring through automatic upgrades and a built-in operations console. See AKS solution journey .

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Kubernetes Engine Monitoring	Azure Monitor container insights	Azure Monitor container insights is a feature designed to monitor the performance of container workloads deployed to: Managed Kubernetes clusters hosted on Azure Kubernetes Service (AKS); Self-managed Kubernetes clusters hosted on Azure using AKS Engine ; Azure Container Instances, Self-managed Kubernetes clusters hosted on Azure Stack or on-premises; or Azure Red Hat OpenShift .
Anthos Service Mesh	Service Fabric Mesh	Fully managed service that enables developers to deploy microservices applications without managing virtual machines, storage, or networking.

Container architectures

Here are some architectures that use AKS as the orchestrator.

- [\[REDACTED\]](#)

Baseline architecture on Azure Kubernetes Service (AKS)

- 07/20/2020
- 37 min read

Deploy a baseline infrastructure that deploys an AKS cluster with focus on security.

- [\[REDACTED\]](#)

Microservices architecture on Azure Kubernetes Service (AKS)

- 5/07/2020
- 17 min read

Deploy a microservices architecture on Azure Kubernetes Service (AKS)

- [\[REDACTED\]](#)

CI/CD pipeline for container-based workloads

- 7/05/2018
- 7 min read

Build a DevOps pipeline for a Node.js web app with Jenkins, Azure Container Registry, Azure Kubernetes Service, Cosmos DB, and Grafana.

[view all](#)

Functions

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
----------------------	---------------	-------------

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Functions	Azure Functions	Integrate systems and run backend processes in response to events or schedules without provisioning or managing servers.

Serverless architectures

- [\[Placeholder\]](#)

Social App for Mobile and Web with Authentication

- 12/16/2019
- 3 min read

View a detailed, step-by-step diagram depicting the build process and implementation of the mobile client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.

- [\[Placeholder\]](#)

HIPAA and HITRUST compliant health data AI

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.

- [\[Placeholder\]](#)

Cross Cloud Scaling Architecture

- 12/16/2019
- 1 min read

Learn how to improve cross cloud scalability with solution architecture that includes Azure Stack. A step-by-step flowchart details instructions for implementation.

DevOps and application monitoring

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Operations (formerly Stackdriver)	Azure Monitor	Maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. It helps you understand how your applications are performing and proactively identifies issues affecting them and the resources on which they depend.

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Trace	Azure Monitor	Maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. It helps you understand how your applications are performing and proactively identifies issues affecting them and the resources on which they depend.
Cloud Debugger	Azure Monitor	Maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. It helps you understand how your applications are performing and proactively identifies issues affecting them and the resources on which they depend.
Cloud Profiler	Azure Monitor	Maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. It helps you understand how your applications are performing and proactively identifies issues affecting them and the resources on which they depend.
Cloud Source Repositories	Azure Repos, GitHub Repos	A cloud service for collaborating on code development.
Cloud Build	Azure Pipelines, GitHub Actions	Fully managed build service that supports continuous integration and deployment.
Artifact Registry	Azure Artifacts, GitHub Packages	Add fully integrated package management to your continuous integration/continuous delivery (CI/CD) pipelines with a single click. Create and share Maven, npm, NuGet, and Python package feeds from public and private sources with teams of any size.
Cloud Developer Tools (including Cloud Code)	Azure Developer Tools	Collection of tools for building, debugging, deploying, diagnosing, and managing multiplatform scalable apps and services.

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Gcloud SDK	Azure CLI	The Azure command-line interface (Azure CLI) is a set of commands used to create and manage Azure resources. The Azure CLI is available across Azure services and is designed to get you working quickly with Azure, with an emphasis on automation.
Cloud Shell	Azure Cloud Shell	Azure Cloud Shell is an interactive, authenticated, browser-accessible shell for managing Azure resources. It provides the flexibility of choosing the shell experience that best suits the way you work, either Bash or PowerShell.
PowerShell on Google Cloud	Azure PowerShell	Azure PowerShell is a set of cmdlets for managing Azure resources directly from the PowerShell command line. Azure PowerShell is designed to make it easy to learn and get started with, but provides powerful features for automation. Written in .NET Standard, Azure PowerShell works with PowerShell 5.1 on Windows, and PowerShell 6.x and higher on all platforms.
Cloud Deployment Manager	Azure Automation	Delivers a cloud-based automation and configuration service that supports consistent management across your Azure and non-Azure environments. It comprises process automation, configuration management, update management, shared capabilities, and heterogeneous features. Automation gives you complete control during deployment, operations, and decommissioning of workloads and resources.
Cloud Deployment Manager	Azure Resource Manager	Provides a way for users to automate the manual, long-running, error-prone, and frequently repeated IT tasks.

DevOps architectures

-

Container CI/CD using Jenkins and Kubernetes on Azure Kubernetes Service (AKS)

- 12/16/2019
- 2 min read

Containers make it easy for you to continuously build and deploy applications. By orchestrating the deployment of those containers using Azure Kubernetes Service (AKS), you can achieve replicable, manageable clusters of containers.

-

Run a Jenkins server on Azure

- 11/19/2020
- 6 min read

Recommended architecture that shows how to deploy and operate a scalable, enterprise-grade Jenkins server on Azure secured with single sign-on (SSO).

- [Architecture](#)

DevOps in a hybrid environment

- 12/16/2019
- 3 min read

The tools provided in Azure allow for the implementation of a DevOps strategy that capably manages both cloud and on-premises environments in tandem.

[view all](#)

Internet of things (IoT)

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud IoT Core	Azure IoT Hub , Azure Event Hubs	A cloud gateway for managing bidirectional communication with billions of IoT devices, securely and at scale.
Cloud Pub/Sub	Azure Stream Analytics , HDInsight Kafka	Process and route streaming data to a subsequent processing engine or to a storage or database platform.
Edge TPU	Azure IoT Edge	Deploy cloud intelligence directly on IoT devices to run in on-premises scenarios.

IOT architectures

- [Architecture](#)

IoT Architecture Azure IoT Subsystems

- 12/16/2019
- 1 min read

Learn about our recommended IoT application architecture that supports hybrid cloud and edge computing. A flowchart details how the subsystems function within the IoT application.

- [Architecture](#)

[Azure IoT reference architecture](#)

- 9/10/2020
- 12 min read

Recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components



Process real-time vehicle data using IoT

- 11/17/2020
- 5 min read

This example builds a real-time data ingestion/processing pipeline to ingest and process messages from IoT devices into a big data analytic platform in Azure.

[view all](#)

Management

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Billing	Azure Billing API	Services to help generate, monitor, forecast, and share billing data for resource usage by time, organization, or product resources.
Cloud Console	Azure portal	A unified management console that simplifies building, deploying, and operating your cloud resources.
Operations (formerly Stackdriver)	Azure Monitor	Comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.
Cost Management	Azure Cost Management	Azure Cost Management helps you understand your Azure invoice, manage your billing account and subscriptions, control Azure spending, and optimize resource use.

Messaging and eventing

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Pub/Sub	Azure Service Bus	Supports a set of cloud-based, message-oriented middleware technologies including reliable message queuing and durable publish/subscribe messaging.
Cloud Pub/Sub	Azure Event Grid	A fully managed event routing service that allows for uniform event consumption using a publish/subscribe model.
Cloud Pub/Sub	Azure Event Hubs	A real-time data ingestion and microbatching service used to build dynamic data pipelines and integrates with other Azure services.

Messaging architectures

- [REDACTED]

Anomaly Detector Process

- 12/16/2019
- 1 min read

Learn more about Anomaly Detector with a step-by-step flowchart that details the process. See how anomaly detection models are selected with time-series data.

- [REDACTED]

Scalable web application

- 10/03/2019
- 7 min read

Use the proven practices in this reference architecture to improve scalability and performance in an Azure App Service web application..

- [REDACTED]

Enterprise integration using queues and events

- 12/03/2018
- 5 min read

Recommended architecture for implementing an enterprise integration pattern with Azure Logic Apps, Azure API Management, Azure Service Bus, and Azure Event Grid.

- [REDACTED]

Ops automation using Event Grid

- 12/16/2019
- 1 min read

Event Grid allows you to speed automation and simplify policy enforcement. For example, Event Grid can notify Azure Automation when a virtual machine is created, or a SQL Database is spun up. These events can be used to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, or file work items.

Networking

AREA	GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
------	----------------------	---------------	-------------

Area	Google Cloud Service	Azure Service	Description
Cloud virtual networking	Virtual Private Network (VPC)	Azure Virtual Network (Vnet)	Provides an isolated, private environment in the cloud. Users have control over their virtual networking environment, including selection of their own IP address range, adding/updating address ranges, creation of subnets, and configuration of route tables and network gateways.
DNS management	Cloud DNS	Azure DNS	Manage your DNS records using the same credentials that are used for billing and support contract as your other Azure services
	Cloud DNS	Azure Traffic Manager	Azure Traffic Manager is a DNS-based load balancer that enables you to distribute traffic optimally to services across global Azure regions, while providing high availability and responsiveness.
	Internal DNS	Azure Private DNS	Manages and resolves domain names in the virtual network, without the need to configure a custom DNS solution, and it provides a naming resolution for virtual machines (VMs) within a virtual network and any connected virtual networks.
Hybrid Connectivity	Cloud Interconnect	Azure ExpressRoute	Establishes a private network connection from a location to the cloud provider (not over the Internet).
	Cloud VPN Gateway	Azure Virtual Network Gateway	Connects Azure virtual networks to other Azure virtual networks, or customer on-premises networks (site-to-site). Allows end users to connect to Azure services through VPN tunneling (point-to-site).
	Cloud VPN Gateway	Azure Virtual WAN	Azure virtual WAN simplifies large-scale branch connectivity with VPN and ExpressRoute.

Area	Google Cloud Service	Azure Service	Description
	Cloud router	Azure Virtual Network Gateway	Enables dynamic routes exchange using BGP.
Load balancing	Network Load Balancing	Azure Load Balancer	Azure Load Balancer load-balances traffic at layer 4 (all TCP or UDP).
	Global load balancing	Azure Front door	Azure front door enables global load balancing across regions using a single anycast IP.
	Global load balancing	Azure Application Gateway	Application Gateway is a layer 7 load balancer. It takes backends with any IP that is reachable. It supports SSL termination, cookie-based session affinity, and round robin for load-balancing traffic.
	Global load balancing	Azure Traffic Manager	Azure Traffic Manager is a DNS-based load balancer that enables you to distribute traffic optimally to services across global Azure regions, while providing high availability and responsiveness.
Content delivery network	Cloud CDN	Azure CDN	A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users.
Firewall	Firewall rules	Application security groups	Azure Application security groups allow you to group virtual machines and define network security policies based on those groups.
	Firewall rules	Network Security groups	Azure network security group filters network traffic to and from Azure resources in an Azure virtual network.
	Firewall rules	Azure Firewall	Azure Firewall is a managed, cloud-based network security service that protects your Azure Virtual Network resources. It's a fully stateful firewall as a service with built-in high availability and unrestricted cloud scalability.

Area	Google Cloud Service	Azure Service	Description
Web Application Firewall	Cloud Armor	Application Gateway - Web Application Firewall	Azure Web Application Firewall (WAF) provides centralized protection of your web applications from common exploits and vulnerabilities.
	Cloud Armor	Front door – Azure Web Application Firewall	Azure Web Application Firewall (WAF) on Azure Front Door provides centralized protection for your web applications.
	Cloud Armor	CDN – Azure Web Application Firewall	Azure Web Application Firewall (WAF) on Azure Content Delivery Network (CDN) from Microsoft provides centralized protection for your web content.
NAT Gateway	Cloud NAT	Azure Virtual Network NAT	Virtual Network NAT (network address translation) provides outbound NAT translations for internet connectivity for virtual networks.
Private Connectivity to PaaS	VPC Service controls	Azure Private Link	Azure Private Link enables you to access Azure PaaS Services and Azure hosted customer-owned/partner services over a private endpoint in your virtual network.
Telemetry	VPC Flow logs	NSG Flow logs	Network security group (NSG) flow logs are a feature of Network Watcher that allows you to view information about ingress and egress IP traffic through an NSG.
	Firewall Rules Logging	NSG Flow logs	Network security group (NSG) flow logs are a feature of Network Watcher that allows you to view information about ingress and egress IP traffic through an NSG.

Area	Google Cloud Service	Azure Service	Description
	Operations (formerly Stackdriver)	Azure Monitor	Azure Monitor delivers a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. Log queries help you maximize the value of the data collected in Azure Monitor Logs.
	Network Intelligence Center	Azure Network Watcher	Azure Network Watcher provides tools to monitor, diagnose, view metrics, and enable or disable logs for resources in an Azure virtual network.
Other Connectivity Options	S2S,P2S	Direct Interconnect,Partner Interconnect,Carrier Peering	Point to Site lets you create a secure connection to your virtual network from an individual client computer. Site to Site is a connection between two or more networks, such as a corporate network and a branch office network.

Networking architectures

- [\[REDACTED\]](#)

Deploy highly available NVAs

- 12/08/2018
- 7 min read

Learn how to deploy network virtual appliances for high availability in Azure. This article includes example architectures for ingress, egress, and both.

- [\[REDACTED\]](#)

Hub-spoke network topology in Azure

- 9/30/2020
- 7 min read

Learn how to implement a hub-spoke topology in Azure, where the hub is a virtual network and the spokes are virtual networks that peer with the hub.

- [\[REDACTED\]](#)

Implement a secure hybrid network

- 1/07/2020
- 9 min read

See a secure hybrid network that extends an on-premises network to Azure with a perimeter network

between the on-premises network and an Azure virtual network.

[view all](#)

Security and identity

Area	Google Cloud Service	Azure Service	Description
Authentication and authorization	Cloud Identity	Azure Active Directory	The Azure Active Directory (Azure AD) enterprise identity service provides single sign-on and multi-factor authentication, which enable the central management of users/groups and external identities federation.
	Identity platform	Azure Active Directory B2C	A highly available and global identity management service for consumer-facing applications, which scales to hundreds of millions of identities. Manage customer, consumer, and citizen access to your business-to-consumer (B2C) applications.
Multi-factor Authentication	Multi-factor Authentication	Azure Active Directory Multi-factor Authentication	Safeguard access to data and applications, while meeting user demand for a simple sign-in process.
RBAC	Identity and Access Management	Azure role-based access control	Azure role-based access control (Azure RBAC) helps you manage who has access to Azure resources, what they can do with those resources, and what areas they have access to.
ABAC	Identity and Access Management	Azure attribute-based access control	Azure attribute-based access control (Azure ABAC) is an authorization system that defines access, based on attributes that are associated with security principals, resources, and environment.
Zero trust	BeyondCorp Enterprise	Azure AD Conditional Access	Conditional Access is the tool used by Azure Active Directory to bring signals together, to make decisions, and to enforce organizational policies.

Area	Google Cloud Service	Azure Service	Description
Resource management	Resource Manager	Azure Resource Manager	Provides a management layer that enables you to create, update, and delete resources in your Azure account, like access control, locks, and tags, to secure and organize your resources after deployment.
Encryption	Cloud KMS , Secret Manager	Azure Key Vault	Provides a security solution and works with other services by allowing you to manage, create, and control encryption keys that are stored in hardware security modules (HSM).
Data-at-rest encryption	Encryption at rest	Azure Storage Service Encryption - encryption by default	Azure Storage Service Encryption helps you protect and safeguard your data and meet your organizational security and compliance commitments.
Data in-use	Confidential Computing	Azure Confidential Computing	Encrypt data in-use.
Hardware security module (HSM)	Cloud HSM	Azure Dedicated HSM	Azure service that provides cryptographic key storage in Azure, to host encryption keys and perform cryptographic operations in a high-availability service of FIPS 140-2 Level 3 certified hardware security modules (HSMs).
Data loss prevention (DLP)	Cloud Data Loss Prevention	Azure Information Protection	Azure Information Protection (AIP) is a cloud-based solution that enables organizations to discover, classify, and protect documents and emails by applying labels to content.
Security	Security Command Center , Web Security Scanner	Microsoft Defender for Cloud	An automated security assessment service that improves the security and compliance of applications. Automatically assess applications for vulnerabilities or deviations from best practices.
Threat detection	Event Threat Detection	Azure Advanced Threat Protection	Detect and investigate advanced attacks on-premises and in the cloud.

Area	Google Cloud Service	Azure Service	Description
SIEM	Chronicle	Microsoft Sentinel	A cloud-native security information and event manager (SIEM) platform that uses built-in AI to help analyze large volumes of data from all sources, including users, applications, servers, and devices that are running on-premises or in any cloud.
Container security	Container Security	Container Security in Microsoft Defender for Cloud	Microsoft Defender for Cloud is the Azure-native solution for securing your containers.
	Artifact Registry	Azure Container Registry	A managed, private Docker registry service that's based on the open-source Docker Registry 2.0. Create and maintain Azure container registries to store and manage your private Docker container images and related artifacts that allow you to only deploy trusted containers.
	Container Analysis	Microsoft Defender for container registries	Perform vulnerability scans on all container images when they're pushed to the registry, imported into the registry, or pulled within the last 30 days.

Security architectures

- [\[link\]](#)

Real-time fraud detection

- 7/05/2018
- 4 min read

Detect fraudulent activity in real-time using Azure Event Hubs and Stream Analytics.

- [\[link\]](#)

Securely managed web applications

- 5/09/2019
- 8 min read

Learn about deploying secure applications using the Azure App Service Environment, the Azure Application Gateway service, and Web Application Firewall.

- [\[link\]](#)

Threat indicators for cyber threat intelligence in Azure Sentinel

- 4/13/2020
- 13 min read

Import threat indicators, view logs, create rules to generate security alerts and incidents, and visualize threat intelligence data with Azure Sentinel.

[view all](#)

Storage

Object storage

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Cloud Storage	Azure Blob storage	Object storage service, for use cases including cloud applications, content distribution, backup, archiving, disaster recovery, and big data analytics.
Cloud Storage for Firebase		

Block storage

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Persistant Disk	Azure managed disks	SSD storage optimized for I/O intensive read/write operations. For use as high-performance Azure virtual machine storage.
Local SSD		

File storage

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Filestore	Azure Files, Azure NetApp Files	File based storage and hosted NetApp Appliance Storage.
Google Drive	OneDrive For business	Cloud storage and file sharing solution for businesses to store, access, and share files anytime and anywhere.

Storage architectures

-

HIPAA and HITRUST compliant health data AI

- 12/16/2019
- 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.

-

HPC Media Rendering

- 11/04/2020
- 2 min read

Optimize the media rendering process with a step-by-step HPC solution architecture from Azure that combines Azure CycleCloud and HPC Cache.

- [redacted]

[Medical Data Storage Solutions](#)

- 12/16/2019
- 2 min read

Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security.

[view all](#)

Bulk data transfer

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
Transfer Appliance	Azure Import/Export	A data transport solution that uses secure disks and appliances to transfer large amounts of data. Also offers data protection during transit.
Transfer Appliance	Azure Data Box	Petabyte- to exabyte-scale data transport solution that uses secure data storage devices to transfer large amounts of data to and from Azure.

Application services

GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
App Engine	Azure App Service	Managed hosting platform providing easy to use services for deploying and scaling web applications and services.
Apigee	Azure API Management	A turnkey solution for publishing APIs to external and internal consumers.

Web architectures

- [redacted]

[Architect scalable e-commerce web app](#)

- 12/16/2019
- 1 min read

The e-commerce website includes simple order processing workflows with the help of Azure services. Using Azure Functions and Web Apps, developers can focus on building personalized experiences and let Azure take care of the infrastructure.

- [redacted]

Multi-region N-tier application

- 6/18/2019
- 10 min read

Deploy an application on Azure virtual machines in multiple regions for high availability and resiliency.

- [View details](#)

Serverless web application

- 5/28/2019
- 16 min read

This reference architecture shows a serverless web application, which serves static content from Azure Blob Storage and implements an API using Azure Functions.

[View all](#)

Miscellaneous

Area	Google Cloud Service	Azure Service	Description
Workflow	Composer	Azure Logic Apps	Serverless technology for connecting apps, data and devices anywhere, whether on-premises or in the cloud for large ecosystems of SaaS and cloud-based connectors.
Enterprise application services	G Suite	Microsoft 365	Fully integrated Cloud service providing communications, email, document management in the cloud and available on a wide variety of devices.
Gaming	Game Servers	Azure PlayFab	Managed services for hosting dedicated game servers.
Hybrid	Anthos	Azure Arc	For customers who want to simplify complex and distributed environments across on-premises, edge and multi-cloud, Azure Arc enables deployment of Azure services anywhere and extends Azure management to any infrastructure.
Blockchain	Digital Asset	Azure Confidential Ledger	Tamperproof, unstructured data store hosted in trusted execution environments and backed by cryptographically verifiable evidence.

Area	Google Cloud Service	Azure Service	Description
Monitoring	Cloud Monitoring	Application Insights	Service that provides visibility into the performance, uptime, and overall health of cloud-powered applications.
Logging	Cloud Logging	Log Analytics	Service for real-time log management and analysis.

Migration tools

Area	Google Cloud Service	Azure Service	Description
App migration to containers	Migrate for Anthos	Azure Migrate: App Containerization tool	Modernize your application by migrating it to AKS or App Services containers.
Migration of virtual machines	Migrate for Compute Engine	Azure Migrate: Server Migration tool	Migrate servers from anywhere to Azure.
VMware migration	Google Cloud VMware Engine	Azure VMware Solution	Move or extend on-premises VMware environments to Azure.
Migration of databases	Database Migration Service	Azure Database Migration Service	Fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms with minimal downtime.
Migration programs	Google Cloud Rapid Assessment & Migration Program (RAMP)	Azure Migration and Modernization Program	Learn how to move your apps, data, and infrastructure to Azure using a proven cloud migration and modernization approach.
Server assessment		Movere	Increases business intelligence by accurately presenting entire IT environments within a single day.
Database assessment		Data Migration Assistant	It helps pinpoint potential problems blocking migration. It identifies unsupported features, new features that can benefit you after migration, and the right path for database migration.
Web app assessment and migration		Web app migration assistant	Assess on-premises web apps and migrate them to Azure.

AREA	GOOGLE CLOUD SERVICE	AZURE SERVICE	DESCRIPTION
------	----------------------	---------------	-------------

More learning

If you are new to Azure, review the interactive [Core Cloud Services - Introduction to Azure](#) module on [Microsoft Learn](#).

Cloud Design Patterns

3/10/2022 • 5 minutes to read • [Edit Online](#)

These design patterns are useful for building reliable, scalable, secure applications in the cloud.

Each pattern describes the problem that the pattern addresses, considerations for applying the pattern, and an example based on Microsoft Azure. Most of the patterns include code samples or snippets that show how to implement the pattern on Azure. However, most of the patterns are relevant to any distributed system, whether hosted on Azure or on other cloud platforms.

Challenges in cloud development



Data Management

Data management is the key element of cloud applications, and influences most of the quality attributes. Data is typically hosted in different locations and across multiple servers for reasons such as performance, scalability or availability, and this can present a range of challenges. For example, data consistency must be maintained, and data will typically need to be synchronized across different locations.



Design and Implementation

Good design encompasses factors such as consistency and coherence in component design and deployment, maintainability to simplify administration and development, and reusability to allow components and subsystems to be used in other applications and in other scenarios. Decisions made during the design and implementation phase have a huge impact on the quality and the total cost of ownership of cloud hosted applications and services.



Messaging

The distributed nature of cloud applications requires a messaging infrastructure that connects the components and services, ideally in a loosely coupled manner in order to maximize scalability. Asynchronous messaging is widely used, and provides many benefits, but also brings challenges such as the ordering of messages, poison message management, idempotency, and more.

Catalog of patterns

PATTERN	SUMMARY	CATEGORY
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.	Design and Implementation , Operational Excellence

PATTERN	SUMMARY	CATEGORY
Anti-Corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.	Design and Implementation, Operational Excellence
Asynchronous Request-Reply	Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.	Messaging
Backends for Frontends	Create separate backend services to be consumed by specific frontend applications or interfaces.	Design and Implementation
Bulkhead	Isolate elements of an application into pools so that if one fails, the others will continue to function.	Reliability
Cache-Aside	Load data on demand into a cache from a data store	Data Management, Performance Efficiency
Choreography	Let each service decide when and how a business operation is processed, instead of depending on a central orchestrator.	Messaging, Performance Efficiency
Circuit Breaker	Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.	Reliability
Claim Check	Split a large message into a claim check and a payload to avoid overwhelming a message bus.	Messaging
Compensating Transaction	Undo the work performed by a series of steps, which together define an eventually consistent operation.	Reliability
Competing Consumers	Enable multiple concurrent consumers to process messages received on the same messaging channel.	Messaging
Compute Resource Consolidation	Consolidate multiple tasks or operations into a single computational unit	Design and Implementation
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.	Data Management, Design and Implementation, Performance Efficiency
Deployment Stamps	Deploy multiple independent copies of application components, including data stores.	Reliability, Performance Efficiency

PATTERN	SUMMARY	CATEGORY
Event Sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.	Data Management, Performance Efficiency
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.	Design and Implementation, Operational Excellence
Federated Identity	Delegate authentication to an external identity provider.	Security
Gatekeeper	Protect applications and services by using a dedicated host instance that acts as a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them.	Security
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.	Design and Implementation, Operational Excellence
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.	Design and Implementation, Operational Excellence
Gateway Routing	Route requests to multiple services using a single endpoint.	Design and Implementation, Operational Excellence
Geodes	Deploy backend services into a set of geographical nodes, each of which can service any client request in any region.	Reliability, Operational Excellence
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.	Reliability, Operational Excellence
Index Table	Create indexes over the fields in data stores that are frequently referenced by queries.	Data Management, Performance Efficiency
Leader Election	Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the other instances.	Design and Implementation, Reliability
Materialized View	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for required query operations.	Data Management, Operational Excellence

PATTERN	SUMMARY	CATEGORY
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.	Design and Implementation, Messaging
Priority Queue	Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.	Messaging, Performance Efficiency
Publisher/Subscriber	Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers.	Messaging
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads.	Reliability, Messaging, Resiliency, Performance Efficiency
Retry	Enable an application to handle anticipated, temporary failures when it tries to connect to a service or network resource by transparently retrying an operation that's previously failed.	Reliability
Scheduler Agent Supervisor	Coordinate a set of actions across a distributed set of services and other remote resources.	Messaging, Reliability
Sequential Convoy	Process a set of related messages in a defined order, without blocking processing of other groups of messages.	Messaging
Sharding	Divide a data store into a set of horizontal partitions or shards.	Data Management, Performance Efficiency
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.	Design and Implementation, Operational Excellence
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.	Design and Implementation, Data Management, Performance Efficiency
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.	Design and Implementation, Operational Excellence

PATTERN	SUMMARY	CATEGORY
Throttling	Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service.	Reliability , Performance Efficiency
Valet Key	Use a token or key that provides clients with restricted direct access to a specific resource or service.	Data Management , Security

Data management patterns

3/10/2022 • 2 minutes to read • [Edit Online](#)

Data management is the key element of cloud applications, and influences most of the quality attributes. Data is typically hosted in different locations and across multiple servers for reasons such as performance, scalability or availability, and this can present a range of challenges. For example, data consistency must be maintained, and data will typically need to be synchronized across different locations.

Additionally data should be protected at rest, in transit, and via authorized access mechanisms to maintain security assurances of confidentiality, integrity, and availability. Refer to the Azure Security Benchmark [Data Protection Control](#) for more information.

PATTERN	SUMMARY
Cache-Aside	Load data on demand into a cache from a data store
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Event Sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.
Index Table	Create indexes over the fields in data stores that are frequently referenced by queries.
Materialized View	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for required query operations.
Sharding	Divide a data store into a set of horizontal partitions or shards.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.
Valet Key	Use a token or key that provides clients with restricted direct access to a specific resource or service.

Design and implementation patterns

3/10/2022 • 2 minutes to read • [Edit Online](#)

Good design encompasses factors such as consistency and coherence in component design and deployment, maintainability to simplify administration and development, and reusability to allow components and subsystems to be used in other applications and in other scenarios. Decisions made during the design and implementation phase have a huge impact on the quality and the total cost of ownership of cloud hosted applications and services.

PATTERN	SUMMARY
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.
Anti-Corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.
Backends for Frontends	Create separate backend services to be consumed by specific frontend applications or interfaces.
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Compute Resource Consolidation	Consolidate multiple tasks or operations into a single computational unit
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.
Gateway Routing	Route requests to multiple services using a single endpoint.
Leader Election	Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the other instances.
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.

PATTERN	SUMMARY
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

Messaging patterns

3/10/2022 • 2 minutes to read • [Edit Online](#)

The distributed nature of cloud applications requires a messaging infrastructure that connects the components and services, ideally in a loosely coupled manner in order to maximize scalability. Asynchronous messaging is widely used, and provides many benefits, but also brings challenges such as the ordering of messages, poison message management, idempotency, and more.

PATTERN	SUMMARY
Asynchronous Request-Reply	Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.
Claim Check	Split a large message into a claim check and a payload to avoid overwhelming a message bus.
Choreography	Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.
Competing Consumers	Enable multiple concurrent consumers to process messages received on the same messaging channel.
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.
Priority Queue	Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.
Publisher-Subscriber	Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers.
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads.
Scheduler Agent Supervisor	Coordinate a set of actions across a distributed set of services and other remote resources.
Sequential Convoy	Process a set of related messages in a defined order, without blocking processing of other groups of messages.

Ambassador pattern

3/10/2022 • 3 minutes to read • [Edit Online](#)

Create helper services that send network requests on behalf of a consumer service or application. An ambassador service can be thought of as an out-of-process proxy that is co-located with the client.

This pattern can be useful for offloading common client connectivity tasks such as monitoring, logging, routing, security (such as TLS), and [resiliency patterns](#) in a language agnostic way. It is often used with legacy applications, or other applications that are difficult to modify, in order to extend their networking capabilities. It can also enable a specialized team to implement those features.

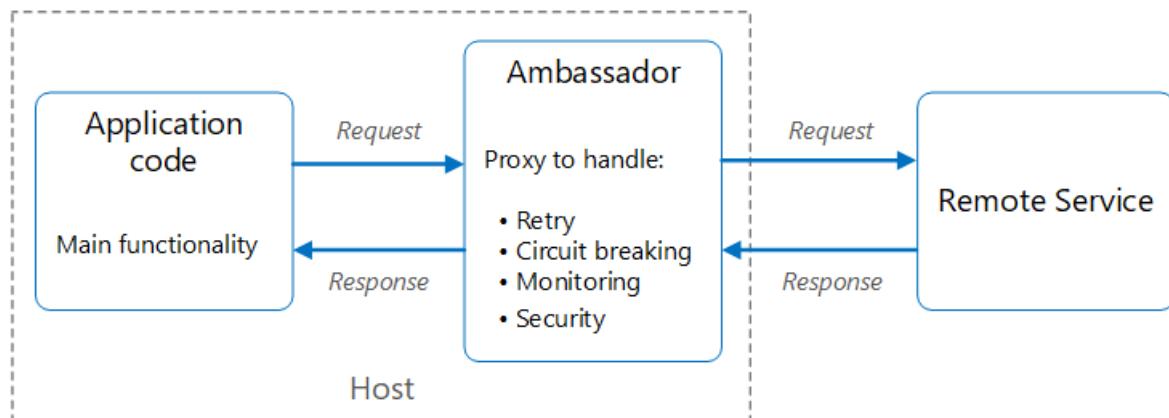
Context and problem

Resilient cloud-based applications require features such as [circuit breaking](#), routing, metering and monitoring, and the ability to make network-related configuration updates. It may be difficult or impossible to update legacy applications or existing code libraries to add these features, because the code is no longer maintained or can't be easily modified by the development team.

Network calls may also require substantial configuration for connection, authentication, and authorization. If these calls are used across multiple applications, built using multiple languages and frameworks, the calls must be configured for each of these instances. In addition, network and security functionality may need to be managed by a central team within your organization. With a large code base, it can be risky for that team to update application code they aren't familiar with.

Solution

Put client frameworks and libraries into an external process that acts as a proxy between your application and external services. Deploy the proxy on the same host environment as your application to allow control over routing, resiliency, security features, and to avoid any host-related access restrictions. You can also use the ambassador pattern to standardize and extend instrumentation. The proxy can monitor performance metrics such as latency or resource usage, and this monitoring happens in the same host environment as the application.



Features that are offloaded to the ambassador can be managed independently of the application. You can update and modify the ambassador without disturbing the application's legacy functionality. It also allows for separate, specialized teams to implement and maintain security, networking, or authentication features that have been moved to the ambassador.

Ambassador services can be deployed as a [sidecar](#) to accompany the lifecycle of a consuming application or

service. Alternatively, if an ambassador is shared by multiple separate processes on a common host, it can be deployed as a daemon or Windows service. If the consuming service is containerized, the ambassador should be created as a separate container on the same host, with the appropriate links configured for communication.

Issues and considerations

- The proxy adds some latency overhead. Consider whether a client library, invoked directly by the application, is a better approach.
- Consider the possible impact of including generalized features in the proxy. For example, the ambassador could handle retries, but that might not be safe unless all operations are idempotent.
- Consider a mechanism to allow the client to pass some context to the proxy, as well as back to the client. For example, include HTTP request headers to opt out of retry or specify the maximum number of times to retry.
- Consider how you will package and deploy the proxy.
- Consider whether to use a single shared instance for all clients or an instance for each client.

When to use this pattern

Use this pattern when you:

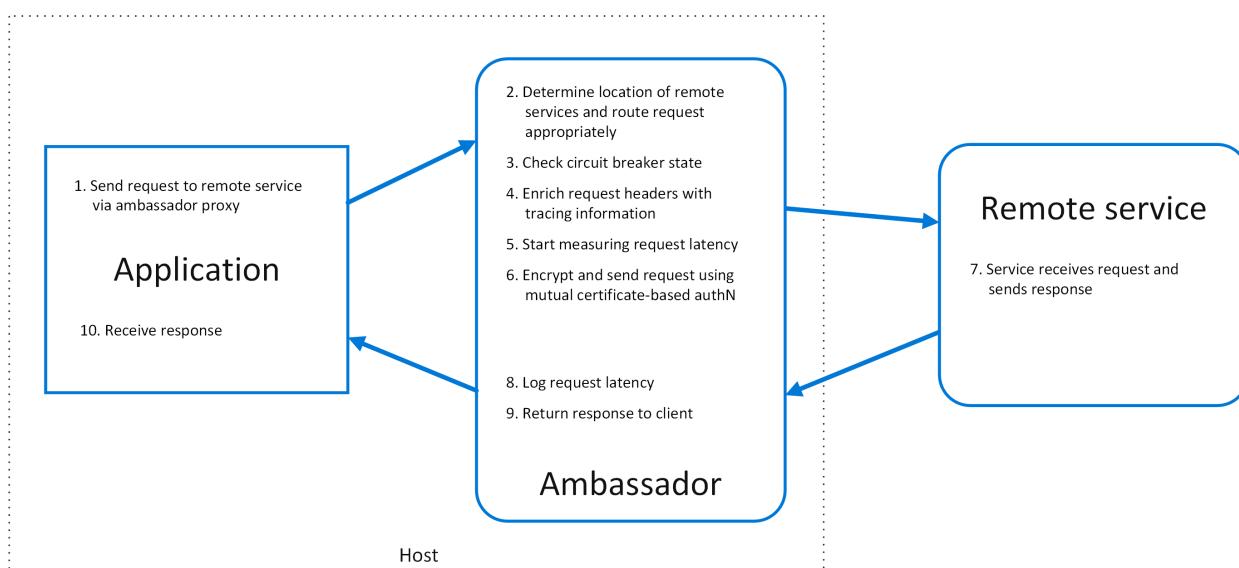
- Need to build a common set of client connectivity features for multiple languages or frameworks.
- Need to offload cross-cutting client connectivity concerns to infrastructure developers or other more specialized teams.
- Need to support cloud or cluster connectivity requirements in a legacy application or an application that is difficult to modify.

This pattern may not be suitable:

- When network request latency is critical. A proxy will introduce some overhead, although minimal, and in some cases this may affect the application.
- When client connectivity features are consumed by a single language. In that case, a better option might be a client library that is distributed to the development teams as a package.
- When connectivity features cannot be generalized and require deeper integration with the client application.

Example

The following diagram shows an application making a request to a remote service via an ambassador proxy. The ambassador provides routing, circuit breaking, and logging. It calls the remote service and then returns the response to the client application:



Related guidance

- [Sidecar pattern](#)

Anti-corruption Layer pattern

3/10/2022 • 2 minutes to read • [Edit Online](#)

Implement a façade or adapter layer between different subsystems that don't share the same semantics. This layer translates requests that one subsystem makes to the other subsystem. Use this pattern to ensure that an application's design is not limited by dependencies on outside subsystems. This pattern was first described by Eric Evans in *Domain-Driven Design*.

Context and problem

Most applications rely on other systems for some data or functionality. For example, when a legacy application is migrated to a modern system, it may still need existing legacy resources. New features must be able to call the legacy system. This is especially true of gradual migrations, where different features of a larger application are moved to a modern system over time.

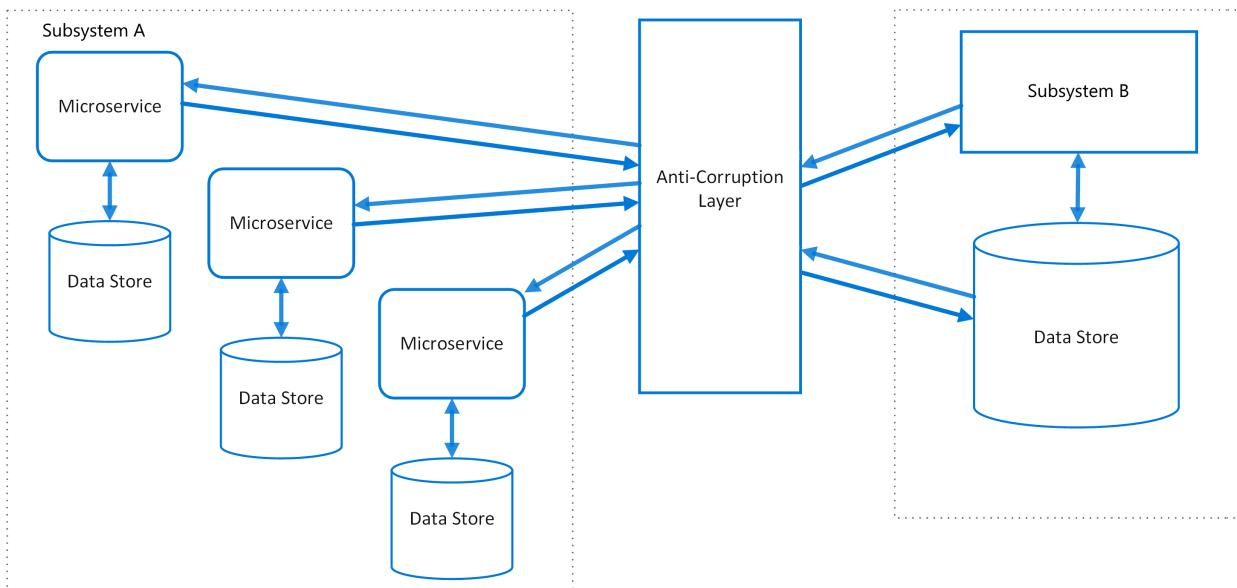
Often these legacy systems suffer from quality issues such as convoluted data schemas or obsolete APIs. The features and technologies used in legacy systems can vary widely from more modern systems. To interoperate with the legacy system, the new application may need to support outdated infrastructure, protocols, data models, APIs, or other features that you wouldn't otherwise put into a modern application.

Maintaining access between new and legacy systems can force the new system to adhere to at least some of the legacy system's APIs or other semantics. When these legacy features have quality issues, supporting them "corrupts" what might otherwise be a cleanly designed modern application.

Similar issues can arise with any external system that your development team doesn't control, not just legacy systems.

Solution

Isolate the different subsystems by placing an anti-corruption layer between them. This layer translates communications between the two systems, allowing one system to remain unchanged while the other can avoid compromising its design and technological approach.



The diagram above shows an application with two subsystems. Subsystem A calls to subsystem B through an anti-corruption layer. Communication between subsystem A and the anti-corruption layer always uses the data

model and architecture of subsystem A. Calls from the anti-corruption layer to subsystem B conform to that subsystem's data model or methods. The anti-corruption layer contains all of the logic necessary to translate between the two systems. The layer can be implemented as a component within the application or as an independent service.

Issues and considerations

- The anti-corruption layer may add latency to calls made between the two systems.
- The anti-corruption layer adds an additional service that must be managed and maintained.
- Consider how your anti-corruption layer will scale.
- Consider whether you need more than one anti-corruption layer. You may want to decompose functionality into multiple services using different technologies or languages, or there may be other reasons to partition the anti-corruption layer.
- Consider how the anti-corruption layer will be managed in relation with your other applications or services. How will it be integrated into your monitoring, release, and configuration processes?
- Make sure transaction and data consistency are maintained and can be monitored.
- Consider whether the anti-corruption layer needs to handle all communication between different subsystems, or just a subset of features.
- If the anti-corruption layer is part of an application migration strategy, consider whether it will be permanent, or will be retired after all legacy functionality has been migrated.

When to use this pattern

Use this pattern when:

- A migration is planned to happen over multiple stages, but integration between new and legacy systems needs to be maintained.
- Two or more subsystems have different semantics, but still need to communicate.

This pattern may not be suitable if there are no significant semantic differences between new and legacy systems.

Related guidance

- [Strangler Fig pattern](#)

Asynchronous Request-Reply pattern

3/10/2022 • 9 minutes to read • [Edit Online](#)

Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.

Context and problem

In modern application development, it's normal for client applications — often code running in a web-client (browser) — to depend on remote APIs to provide business logic and compose functionality. These APIs may be directly related to the application or may be shared services provided by a third party. Commonly these API calls take place over the HTTP(S) protocol and follow REST semantics.

In most cases, APIs for a client application are designed to respond quickly, on the order of 100 ms or less. Many factors can affect the response latency, including:

- An application's hosting stack.
- Security components.
- The relative geographic location of the caller and the backend.
- Network infrastructure.
- Current load.
- The size of the request payload.
- Processing queue length.
- The time for the backend to process the request.

Any of these factors can add latency to the response. Some can be mitigated by scaling out the backend. Others, such as network infrastructure, are largely out of the control of the application developer. Most APIs can respond quickly enough for responses to arrive back over the same connection. Application code can make a synchronous API call in a non-blocking way, giving the appearance of asynchronous processing, which is recommended for I/O-bound operations.

In some scenarios, however, the work done by backend may be long-running, on the order of seconds, or might be a background process that is executed in minutes or even hours. In that case, it isn't feasible to wait for the work to complete before responding to the request. This situation is a potential problem for any synchronous request-reply pattern.

Some architectures solve this problem by using a message broker to separate the request and response stages. This separation is often achieved by use of the [Queue-Based Load Leveling pattern](#). This separation can allow the client process and the backend API to scale independently. But this separation also brings additional complexity when the client requires success notification, as this step needs to become asynchronous.

Many of the same considerations discussed for client applications also apply for server-to-server REST API calls in distributed systems — for example, in a microservices architecture.

Solution

One solution to this problem is to use HTTP polling. Polling is useful to client-side code, as it can be hard to provide call-back endpoints or use long running connections. Even when callbacks are possible, the extra libraries and services that are required can sometimes add too much extra complexity.

- The client application makes a synchronous call to the API, triggering a long-running operation on the

backend.

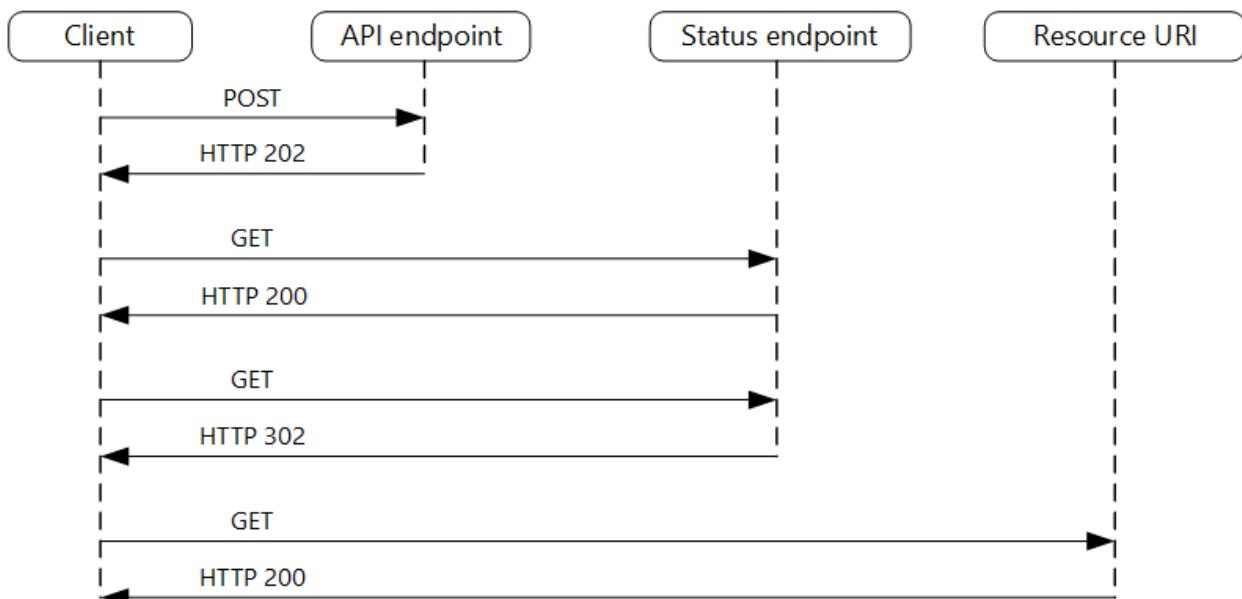
- The API responds synchronously as quickly as possible. It returns an HTTP 202 (Accepted) status code, acknowledging that the request has been received for processing.

NOTE

The API should validate both the request and the action to be performed before starting the long running process. If the request is invalid, reply immediately with an error code such as HTTP 400 (Bad Request).

- The response holds a location reference pointing to an endpoint that the client can poll to check for the result of the long running operation.
- The API offloads processing to another component, such as a message queue.
- For every successful call to the status endpoint, it returns HTTP 202. While the work is still pending, the status endpoint returns a resource that indicates the work is still in progress. Once the work is complete, the status endpoint can either return a resource that indicates completion, or redirect to another resource URL. For example, if the asynchronous operation creates a new resource, the status endpoint would redirect to the URL for that resource.

The following diagram shows a typical flow:



1. The client sends a request and receives an HTTP 202 (Accepted) response.
2. The client sends an HTTP GET request to the status endpoint. The work is still pending, so this call returns HTTP 202.
3. At some point, the work is complete and the status endpoint returns 302 (Found) redirecting to the resource.
4. The client fetches the resource at the specified URL.

Issues and considerations

- There are a number of possible ways to implement this pattern over HTTP and not all upstream services have the same semantics. For example, most services won't return an HTTP 202 response back from a GET method when a remote process hasn't finished. Following pure REST semantics, they should return HTTP 404 (Not Found). This response makes sense when you consider the result of the call isn't present yet.
- An HTTP 202 response should indicate the location and frequency that the client should poll for the response. It should have the following additional headers:

HEADER	DESCRIPTION	NOTES
Location	A URL the client should poll for a response status.	This URL could be a SAS token with the Valet Key Pattern being appropriate if this location needs access control. The valet key pattern is also valid when response polling needs offloading to another backend
Retry-After	An estimate of when processing will complete	This header is designed to prevent polling clients from overwhelming the back-end with retries.

- You may need to use a processing proxy or facade to manipulate the response headers or payload depending on the underlying services used.
- If the status endpoint redirects on completion, either [HTTP 302](#) or [HTTP 303](#) are appropriate return codes, depending on the exact semantics you support.
- Upon successful processing, the resource specified by the Location header should return an appropriate HTTP response code such as 200 (OK), 201 (Created), or 204 (No Content).
- If an error occurs during processing, persist the error at the resource URL described in the Location header and ideally return an appropriate response code to the client from that resource (4xx code).
- Not all solutions will implement this pattern in the same way and some services will include additional or alternate headers. For example, Azure Resource Manager uses a modified variant of this pattern. For more information, see [Azure Resource Manager Async Operations](#).
- Legacy clients might not support this pattern. In that case, you might need to place a facade over the asynchronous API to hide the asynchronous processing from the original client. For example, Azure Logic Apps supports this pattern natively can be used as an integration layer between an asynchronous API and a client that makes synchronous calls. See [Perform long-running tasks with the webhook action pattern](#).
- In some scenarios, you might want to provide a way for clients to cancel a long-running request. In that case, the backend service must support some form of cancellation instruction.

When to use this pattern

Use this pattern for:

- Client-side code, such as browser applications, where it's difficult to provide call-back endpoints, or the use of long-running connections adds too much additional complexity.
- Service calls where only the HTTP protocol is available and the return service can't fire callbacks because of firewall restrictions on the client-side.
- Service calls that need to be integrated with legacy architectures that don't support modern callback technologies such as WebSockets or webhooks.

This pattern might not be suitable when:

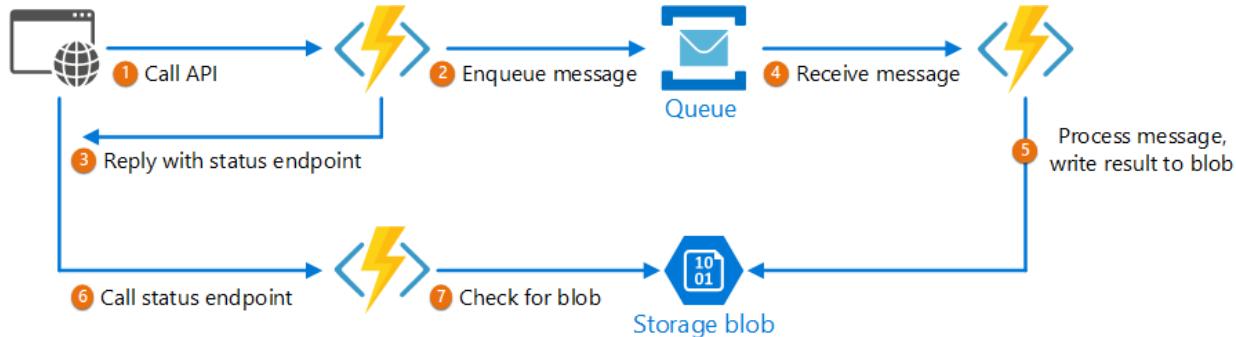
- You can use a service built for asynchronous notifications instead, such as Azure Event Grid.
- Responses must stream in real time to the client.
- The client needs to collect many results, and received latency of those results is important. Consider a service bus pattern instead.

- You can use server-side persistent network connections such as WebSockets or SignalR. These services can be used to notify the caller of the result.
- The network design allows you to open up ports to receive asynchronous callbacks or webhooks.

Example

The following code shows excerpts from an application that uses Azure Functions to implement this pattern. There are three functions in the solution:

- The asynchronous API endpoint.
- The status endpoint.
- A backend function that takes queued work items and executes them.



This sample is available on [GitHub](#).

AsyncProcessingWorkAcceptor function

The `AsyncProcessingWorkAcceptor` function implements an endpoint that accepts work from a client application and puts it on a queue for processing.

- The function generates a request ID and adds it as metadata to the queue message.
- The HTTP response includes a location header pointing to a status endpoint. The request ID is part of the URL path.

```

public static class AsyncProcessingWorkAcceptor
{
    [FunctionName("AsyncProcessingWorkAcceptor")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = null)] CustomerPOCO customer,
        [ServiceBus("outqueue", Connection = "ServiceBusConnectionAppSetting")]
    IAsyncCollector<ServiceBusMessage> OutMessages,
    ILogger log)
    {
        if (String.IsNullOrEmpty(customer.id) || String.IsNullOrEmpty(customer.customername))
        {
            return new BadRequestResult();
        }

        string reqid = Guid.NewGuid().ToString();

        string rqs =
        $"http://{{Environment.GetEnvironmentVariable("WEBSITE_HOSTNAME")}}/api/RequestStatus/{{reqid}}";

        var messagePayload = JsonConvert.SerializeObject(customer);
        var message = new ServiceBusMessage(messagePayload);
        message.ApplicationProperties["RequestGUID"] = reqid;
        message.ApplicationProperties["RequestSubmittedAt"] = DateTime.Now;
        message.ApplicationProperties["RequestStatusURL"] = rqs;

        await OutMessages.AddAsync(message);

        return (ActionResult) new AcceptedResult(rqs, $"Request Accepted for
Processing{{Environment.NewLine}ProxyStatus: {rqs}}");
    }
}

```

AsyncProcessingBackgroundWorker function

The `AsyncProcessingBackgroundWorker` function picks up the operation from the queue, does some work based on the message payload, and writes the result to the SAS signature location.

```

public static class AsyncProcessingBackgroundWorker
{
    [FunctionName("AsyncProcessingBackgroundWorker")]
    public static async Task RunAsync(
        [ServiceBusTrigger("outqueue", Connection = "ServiceBusConnectionAppSetting")] ServiceBusMessage
myQueueItem,
        [Blob("data", FileAccess.ReadWrite, Connection = "StorageConnectionAppSetting")] BlobContainerClient
inputContainer,
        ILogger log)
    {
        // Perform an actual action against the blob data source for the async readers to be able to check
against.
        // This is where your actual service worker processing will be performed.

        var id = myQueueItem.ApplicationProperties["RequestGUID"] as string;

        BlobClient blob = inputContainer.GetBlobClient($"{{id}}.blobdata");

        // Now write the results to blob storage.
        await blob.UploadAsync(myQueueItem.Body);
    }
}

```

AsyncOperationStatusChecker function

The `AsyncOperationStatusChecker` function implements the status endpoint. This function first checks whether the request was completed

- If the request was completed, the function either returns a valet-key to the response, or redirects the call immediately to the valet-key URL.
- If the request is still pending, then we should return a 202 accepted with a self-referencing Location header, putting an ETA for a completed response in the http Retry-After header.

```

public static class AsyncOperationStatusChecker
{
    [FunctionName("AsyncOperationStatusChecker")]
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "RequestStatus/{thisGUID}")] HttpRequest
req,
        [Blob("data/{thisGuid}.blobdata", FileAccess.Read, Connection = "StorageConnectionAppSetting")]
BlockBlobClient inputBlob, string thisGUID,
        ILogger log)
    {

        OnCompleteEnum OnComplete = Enum.Parse<OnCompleteEnum>(req.Query["OnComplete"].FirstOrDefault() ???
"Redirect");
        OnPendingEnum OnPending = Enum.Parse<OnPendingEnum>(req.Query["OnPending"].FirstOrDefault() ???
"Accepted");

        log.LogInformation($"C# HTTP trigger function processed a request for status on {thisGUID} - 
OnComplete {OnComplete} - OnPending {OnPending}");

        // Check to see if the blob is present.
        if (await inputBlob.ExistsAsync())
        {
            // If it's present, depending on the value of the optional "OnComplete" parameter choose what to
do.
            return await OnCompleted(OnComplete, inputBlob, thisGUID);
        }
        else
        {
            // If it's NOT present, check the optional "OnPending" parameter.
            string rqs =
$"http://{Environment.GetEnvironmentVariable("WEBSITE_HOSTNAME")}/api/RequestStatus/{thisGUID}";

            switch (OnPending)
            {
                case OnPendingEnum.Accepted:
                {
                    // Return an HTTP 202 status code.
                    return (ActionResult)new AcceptedResult() { Location = rqs };
                }

                case OnPendingEnum.Synchronous:
                {
                    // Back off and retry. Time out if the backoff period hits one minute
                    int backoff = 250;

                    while (!await inputBlob.ExistsAsync() && backoff < 64000)
                    {
                        log.LogInformation($"Synchronous mode {thisGUID}.blob - retrying in {backoff}
ms");
                        backoff = backoff * 2;
                        await Task.Delay(backoff);
                    }

                    if (await inputBlob.ExistsAsync())
                    {
                        log.LogInformation($"Synchronous Redirect mode {thisGUID}.blob - completed after
{backoff} ms");
                        return await OnCompleted(OnComplete, inputBlob, thisGUID);
                    }
                }
            }
        }
    }
}

```

```

    log.LogInformation($"System thread mode '{thisGUID}.blob' - NOT FOUND after {timeout}
{backoff} ms");
        return (ActionResult)new NotFoundResult();
    }
}

default:
{
    throw new InvalidOperationException($"Unexpected value: {OnPending}");
}
}

private static async Task<IActionResult> OnCompleted(OnCompleteEnum OnComplete, BlockBlobClient
inputBlob, string thisGUID)
{
    switch (OnComplete)
    {
        case OnCompleteEnum.Redirect:
        {
            // Redirect to the SAS URI to blob storage
            return (ActionResult)new RedirectResult(inputBlob.GenerateSASURI());
        }

        case OnCompleteEnum.Stream:
        {
            // Download the file and return it directly to the caller.
            // For larger files, use a stream to minimize RAM usage.
            return (ActionResult)new OkObjectResult(await inputBlob.DownloadContentAsync());
        }

        default:
        {
            throw new InvalidOperationException($"Unexpected value: {OnComplete}");
        }
    }
}

public enum OnCompleteEnum {
    Redirect,
    Stream
}

public enum OnPendingEnum {
    Accepted,
    Synchronous
}

```

Next steps

The following information may be relevant when implementing this pattern:

- [Azure Logic Apps - Perform long-running tasks with the polling action pattern](#).
- For general best practices when designing a web API, see [Web API design](#).

Related guidance

- [Backends for Frontends pattern](#)

Backends for Frontends pattern

3/10/2022 • 3 minutes to read • [Edit Online](#)

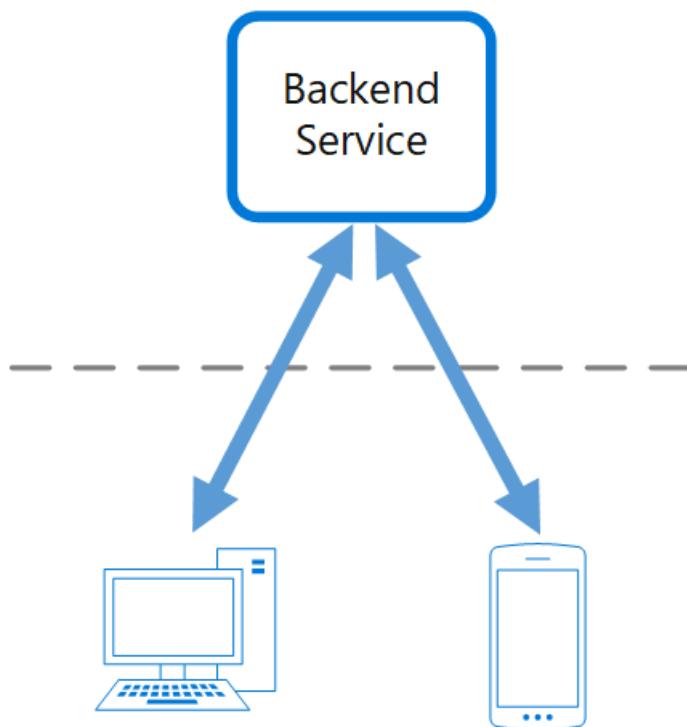
Create separate backend services to be consumed by specific frontend applications or interfaces. This pattern is useful when you want to avoid customizing a single backend for multiple interfaces. This pattern was first described by Sam Newman.

Context and problem

An application may initially be targeted at a desktop web UI. Typically, a backend service is developed in parallel that provides the features needed for that UI. As the application's user base grows, a mobile application is developed that must interact with the same backend. The backend service becomes a general-purpose backend, serving the requirements of both the desktop and mobile interfaces.

But the capabilities of a mobile device differ significantly from a desktop browser, in terms of screen size, performance, and display limitations. As a result, the requirements for a mobile application backend differ from the desktop web UI.

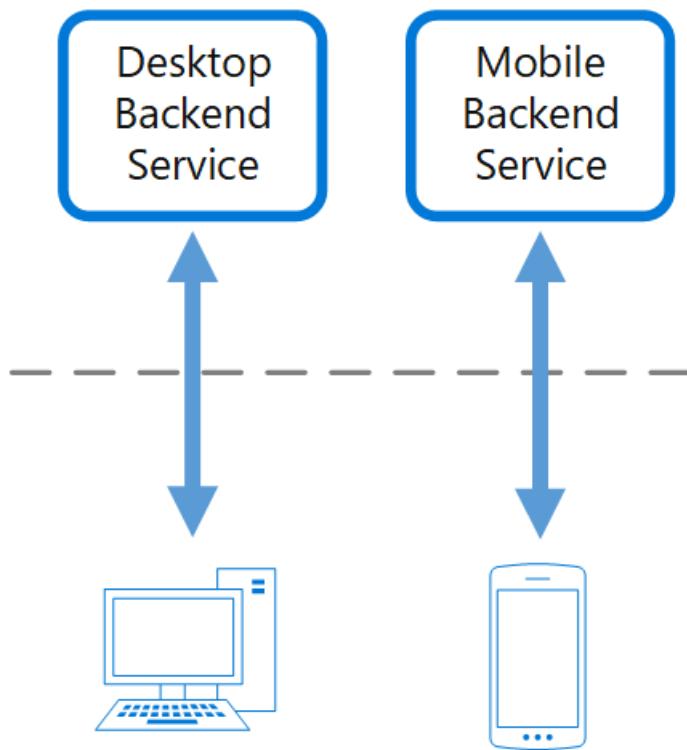
These differences result in competing requirements for the backend. The backend requires regular and significant changes to serve both the desktop web UI and the mobile application. Often, separate interface teams work on each frontend, causing the backend to become a bottleneck in the development process. Conflicting update requirements, and the need to keep the service working for both frontends, can result in spending a lot of effort on a single deployable resource.



As the development activity focuses on the backend service, a separate team may be created to manage and maintain the backend. Ultimately, this results in a disconnect between the interface and backend development teams, placing a burden on the backend team to balance the competing requirements of the different UI teams. When one interface team requires changes to the backend, those changes must be validated with other interface teams before they can be integrated into the backend.

Solution

Create one backend per user interface. Fine-tune the behavior and performance of each backend to best match the needs of the frontend environment, without worrying about affecting other frontend experiences.



Because each backend is specific to one interface, it can be optimized for that interface. As a result, it will be smaller, less complex, and likely faster than a generic backend that tries to satisfy the requirements for all interfaces. Each interface team has autonomy to control their own backend and doesn't rely on a centralized backend development team. This gives the interface team flexibility in language selection, release cadence, prioritization of workload, and feature integration in their backend.

For more information, see [Pattern: Backends For Frontends](#).

Issues and considerations

- Consider how many backends to deploy.
- If different interfaces (such as mobile clients) will make the same requests, consider whether it is necessary to implement a backend for each interface, or if a single backend will suffice.
- Code duplication across services is highly likely when implementing this pattern.
- Frontend-focused backend services should only contain client-specific logic and behavior. General business logic and other global features should be managed elsewhere in your application.
- Think about how this pattern might be reflected in the responsibilities of a development team.
- Consider how long it will take to implement this pattern. Will the effort of building the new backends incur technical debt, while you continue to support the existing generic backend?

When to use this pattern

Use this pattern when:

- A shared or general purpose backend service must be maintained with significant development overhead.
- You want to optimize the backend for the requirements of specific client interfaces.
- Customizations are made to a general-purpose backend to accommodate multiple interfaces.
- An alternative language is better suited for the backend of a different user interface.

This pattern may not be suitable:

- When interfaces make the same or similar requests to the backend.
- When only one interface is used to interact with the backend.

Next steps

- [Pattern: Backends For Frontends](#)

Related guidance

- [Gateway Aggregation pattern](#)
- [Gateway Offloading pattern](#)
- [Gateway Routing pattern](#)

Bulkhead pattern

3/10/2022 • 4 minutes to read • [Edit Online](#)

The Bulkhead pattern is a type of application design that is tolerant of failure. In a bulkhead architecture, elements of an application are isolated into pools so that if one fails, the others will continue to function. It's named after the sectioned partitions (bulkheads) of a ship's hull. If the hull of a ship is compromised, only the damaged section fills with water, which prevents the ship from sinking.

Context and problem

A cloud-based application may include multiple services, with each service having one or more consumers. Excessive load or failure in a service will impact all consumers of the service.

Moreover, a consumer may send requests to multiple services simultaneously, using resources for each request. When the consumer sends a request to a service that is misconfigured or not responding, the resources used by the client's request may not be freed in a timely manner. As requests to the service continue, those resources may be exhausted. For example, the client's connection pool may be exhausted. At that point, requests by the consumer to other services are affected. Eventually the consumer can no longer send requests to other services, not just the original unresponsive service.

The same issue of resource exhaustion affects services with multiple consumers. A large number of requests originating from one client may exhaust available resources in the service. Other consumers are no longer able to consume the service, causing a cascading failure effect.

Solution

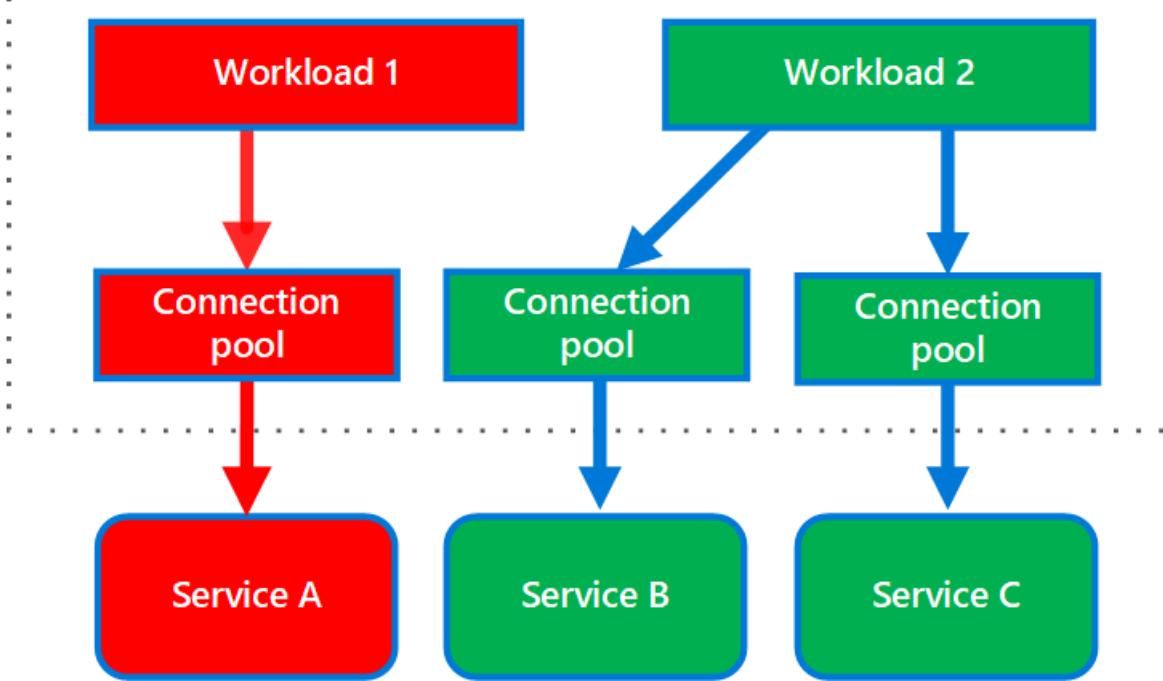
Partition service instances into different groups, based on consumer load and availability requirements. This design helps to isolate failures, and allows you to sustain service functionality for some consumers, even during a failure.

A consumer can also partition resources, to ensure that resources used to call one service don't affect the resources used to call another service. For example, a consumer that calls multiple services may be assigned a connection pool for each service. If a service begins to fail, it only affects the connection pool assigned for that service, allowing the consumer to continue using the other services.

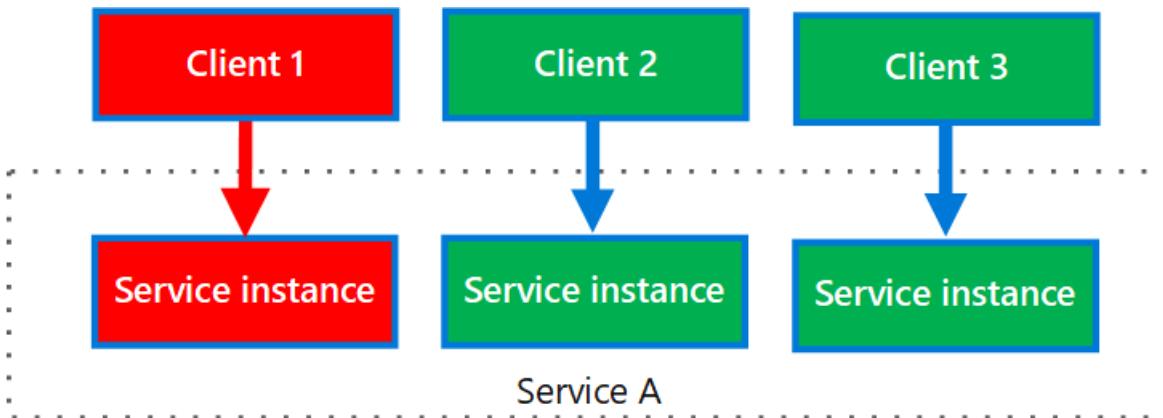
The benefits of this pattern include:

- Isolates consumers and services from cascading failures. An issue affecting a consumer or service can be isolated within its own bulkhead, preventing the entire solution from failing.
- Allows you to preserve some functionality in the event of a service failure. Other services and features of the application will continue to work.
- Allows you to deploy services that offer a different quality of service for consuming applications. A high-priority consumer pool can be configured to use high-priority services.

The following diagram shows bulkheads structured around connection pools that call individual services. If Service A fails or causes some other issue, the connection pool is isolated, so only workloads using the thread pool assigned to Service A are affected. Workloads that use Service B and C are not affected and can continue working without interruption.



The next diagram shows multiple clients calling a single service. Each client is assigned a separate service instance. Client 1 has made too many requests and overwhelmed its instance. Because each service instance is isolated from the others, the other clients can continue making calls.



Issues and considerations

- Define partitions around the business and technical requirements of the application.
- When partitioning services or consumers into bulkheads, consider the level of isolation offered by the technology as well as the overhead in terms of cost, performance and manageability.
- Consider combining bulkheads with retry, circuit breaker, and throttling patterns to provide more sophisticated fault handling.
- When partitioning consumers into bulkheads, consider using processes, thread pools, and semaphores. Projects like [resilience4j](#) and [Polly](#) offer a framework for creating consumer bulkheads.
- When partitioning services into bulkheads, consider deploying them into separate virtual machines, containers, or processes. Containers offer a good balance of resource isolation with fairly low overhead.
- Services that communicate using asynchronous messages can be isolated through different sets of queues. Each queue can have a dedicated set of instances processing messages on the queue, or a single group of instances using an algorithm to dequeue and dispatch processing.
- Determine the level of granularity for the bulkheads. For example, if you want to distribute tenants across partitions, you could place each tenant into a separate partition, or put several tenants into one partition.
- Monitor each partition's performance and SLA.

When to use this pattern

Use this pattern to:

- Isolate resources used to consume a set of backend services, especially if the application can provide some level of functionality even when one of the services is not responding.
- Isolate critical consumers from standard consumers.
- Protect the application from cascading failures.

This pattern may not be suitable when:

- Less efficient use of resources may not be acceptable in the project.
- The added complexity is not necessary

Example

The following Kubernetes configuration file creates an isolated container to run a single service, with its own CPU and memory resources and limits.

```
apiVersion: v1
kind: Pod
metadata:
  name: drone-management
spec:
  containers:
    - name: drone-management-container
      image: drone-service
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "1"
```

Related guidance

- [Designing reliable Azure applications](#)
- [Circuit Breaker pattern](#)
- [Retry pattern](#)
- [Throttling pattern](#)

Cache-Aside pattern

3/10/2022 • 7 minutes to read • [Edit Online](#)

Load data on demand into a cache from a data store. This can improve performance and also helps to maintain consistency between data held in the cache and data in the underlying data store.

Context and problem

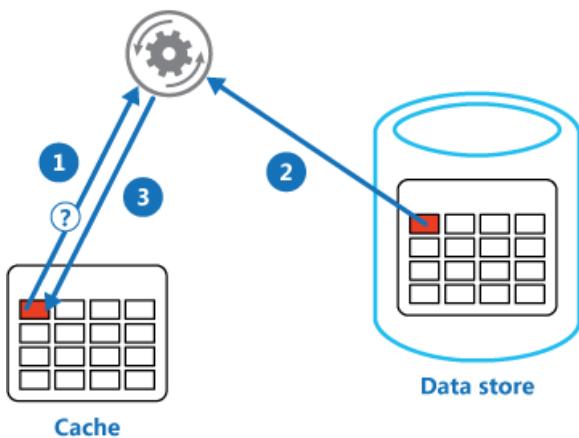
Applications use a cache to improve repeated access to information held in a data store. However, it's impractical to expect that cached data will always be completely consistent with the data in the data store. Applications should implement a strategy that helps to ensure that the data in the cache is as up-to-date as possible, but can also detect and handle situations that arise when the data in the cache has become stale.

Solution

Many commercial caching systems provide read-through and write-through/write-behind operations. In these systems, an application retrieves data by referencing the cache. If the data isn't in the cache, it's retrieved from the data store and added to the cache. Any modifications to data held in the cache are automatically written back to the data store as well.

For caches that don't provide this functionality, it's the responsibility of the applications that use the cache to maintain the data.

An application can emulate the functionality of read-through caching by implementing the cache-aside strategy. This strategy loads data into the cache on demand. The figure illustrates using the Cache-Aside pattern to store data in the cache.



- 1: Determine whether the item is currently held in the cache.
- 2: If the item is not currently in the cache, read the item from the data store.
- 3: Store a copy of the item in the cache.

If an application updates information, it can follow the write-through strategy by making the modification to the data store, and by invalidating the corresponding item in the cache.

When the item is next required, using the cache-aside strategy will cause the updated data to be retrieved from the data store and added back into the cache.

Issues and considerations

Consider the following points when deciding how to implement this pattern:

Lifetime of cached data. Many caches implement an expiration policy that invalidates data and removes it from the cache if it's not accessed for a specified period. For cache-aside to be effective, ensure that the expiration policy matches the pattern of access for applications that use the data. Don't make the expiration period too short because this can cause applications to continually retrieve data from the data store and add it to the cache. Similarly, don't make the expiration period so long that the cached data is likely to become stale. Remember that caching is most effective for relatively static data, or data that is read frequently.

Evicting data. Most caches have a limited size compared to the data store where the data originates, and they'll evict data if necessary. Most caches adopt a least-recently-used policy for selecting items to evict, but this might be customizable. Configure the global expiration property and other properties of the cache, and the expiration property of each cached item, to ensure that the cache is cost effective. It isn't always appropriate to apply a global eviction policy to every item in the cache. For example, if a cached item is very expensive to retrieve from the data store, it can be beneficial to keep this item in the cache at the expense of more frequently accessed but less costly items.

Priming the cache. Many solutions prepopulate the cache with the data that an application is likely to need as part of the startup processing. The Cache-Aside pattern can still be useful if some of this data expires or is evicted.

Consistency. Implementing the Cache-Aside pattern doesn't guarantee consistency between the data store and the cache. An item in the data store can be changed at any time by an external process, and this change might not be reflected in the cache until the next time the item is loaded. In a system that replicates data across data stores, this problem can become serious if synchronization occurs frequently.

Local (in-memory) caching. A cache could be local to an application instance and stored in-memory. Cache-aside can be useful in this environment if an application repeatedly accesses the same data. However, a local cache is private and so different application instances could each have a copy of the same cached data. This data could quickly become inconsistent between caches, so it might be necessary to expire data held in a private cache and refresh it more frequently. In these scenarios, consider investigating the use of a shared or a distributed caching mechanism.

When to use this pattern

Use this pattern when:

- A cache doesn't provide native read-through and write-through operations.
- Resource demand is unpredictable. This pattern enables applications to load data on demand. It makes no assumptions about which data an application will require in advance.

This pattern might not be suitable:

- When the cached data set is static. If the data will fit into the available cache space, prime the cache with the data on startup and apply a policy that prevents the data from expiring.
- For caching session state information in a web application hosted in a web farm. In this environment, you should avoid introducing dependencies based on client-server affinity.

Example

In Microsoft Azure you can use Azure Cache for Redis to create a distributed cache that can be shared by multiple instances of an application.

This following code examples use the [StackExchange.Redis](#) client, which is a Redis client library written for .NET.

To connect to an Azure Cache for Redis instance, call the static `ConnectionMultiplexer.Connect` method and pass in the connection string. The method returns a `ConnectionMultiplexer` that represents the connection. One approach to sharing a `ConnectionMultiplexer` instance in your application is to have a static property that returns a connected instance, similar to the following example. This approach provides a thread-safe way to initialize only a single connected instance.

```
private static ConnectionMultiplexer Connection;

// Redis connection string information
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    string cacheConnection = ConfigurationManager.AppSettings["CacheConnection"].ToString();
    return ConnectionMultiplexer.Connect(cacheConnection);
});

public static ConnectionMultiplexer Connection => lazyConnection.Value;
```

The `GetMyEntityAsync` method in the following code example shows an implementation of the Cache-Aside pattern. This method retrieves an object from the cache using the read-through approach.

An object is identified by using an integer ID as the key. The `GetMyEntityAsync` method tries to retrieve an item with this key from the cache. If a matching item is found, it's returned. If there's no match in the cache, the `GetMyEntityAsync` method retrieves the object from a data store, adds it to the cache, and then returns it. The code that actually reads the data from the data store is not shown here, because it depends on the data store. Note that the cached item is configured to expire to prevent it from becoming stale if it's updated elsewhere.

```
// Set five minute expiration as a default
private const double DefaultExpirationTimeInMinutes = 5.0;

public async Task<MyEntity> GetMyEntityAsync(int id)
{
    // Define a unique key for this method and its parameters.
    var key = $"MyEntity:{id}";
    var cache = Connection.GetDatabase();

    // Try to get the entity from the cache.
    var json = await cache.StringGetAsync(key).ConfigureAwait(false);
    var value = string.IsNullOrWhiteSpace(json)
        ? default(MyEntity)
        : JsonConvert.DeserializeObject<MyEntity>(json);

    if (value == null) // Cache miss
    {
        // If there's a cache miss, get the entity from the original store and cache it.
        // Code has been omitted because it is data store dependent.
        value = ...;

        // Avoid caching a null value.
        if (value != null)
        {
            // Put the item in the cache with a custom expiration time that
            // depends on how critical it is to have stale data.
            await cache.StringSetAsync(key, JsonConvert.SerializeObject(value)).ConfigureAwait(false);
            await cache.KeyExpireAsync(key,
                TimeSpan.FromMinutes(DefaultExpirationTimeInMinutes)).ConfigureAwait(false);
        }
    }

    return value;
}
```

The examples use Azure Cache for Redis to access the store and retrieve information from the cache. For more information, see [Using Azure Cache for Redis](#) and [How to create a Web App with Azure Cache for Redis](#).

The `UpdateEntityAsync` method shown below demonstrates how to invalidate an object in the cache when the value is changed by the application. The code updates the original data store and then removes the cached item from the cache.

```
public async Task UpdateEntityAsync(MyEntity entity)
{
    // Update the object in the original data store.
    await this.store.UpdateEntityAsync(entity).ConfigureAwait(false);

    // Invalidate the current cache object.
    var cache = Connection.GetDatabase();
    var id = entity.Id;
    var key = $"MyEntity:{id}"; // The key for the cached object.
    await cache.KeyDeleteAsync(key).ConfigureAwait(false); // Delete this key from the cache.
}
```

NOTE

The order of the steps is important. Update the data store *before* removing the item from the cache. If you remove the cached item first, there is a small window of time when a client might fetch the item before the data store is updated. That will result in a cache miss (because the item was removed from the cache), causing the earlier version of the item to be fetched from the data store and added back into the cache. The result will be stale cache data.

Related guidance

The following information may be relevant when implementing this pattern:

- [Caching Guidance](#). Provides additional information on how you can cache data in a cloud solution, and the issues that you should consider when you implement a cache.
- [Data Consistency Primer](#). Cloud applications typically use data that's spread across data stores. Managing and maintaining data consistency in this environment is a critical aspect of the system, particularly the concurrency and availability issues that can arise. This primer describes issues about consistency across distributed data, and summarizes how an application can implement eventual consistency to maintain the availability of data.

Choreography pattern

3/10/2022 • 7 minutes to read • [Edit Online](#)

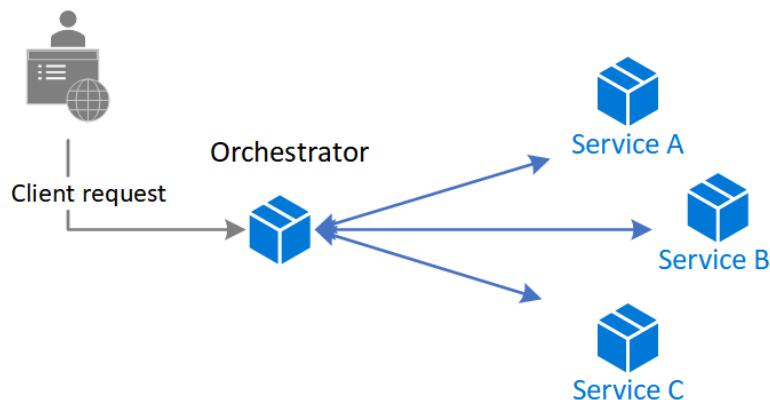
Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.

Context and problem

In microservices architecture, it's often the case that a cloud-based application is divided into several small services that work together to process a business transaction end-to-end. To lower coupling between services, each service is responsible for a single business operation. Some benefits include faster development, smaller code base, and scalability. However, designing an efficient and scalable workflow is a challenge and often requires complex interservice communication.

The services communicate with each other by using well-defined APIs. Even a single business operation can result in multiple point-to-point calls among all services. A common pattern for communication is to use a centralized service that acts as the orchestrator. It acknowledges all incoming requests and delegates operations to the respective services. In doing so, it also manages the workflow of the entire business transaction. Each service just completes an operation and is not aware of the overall workflow.

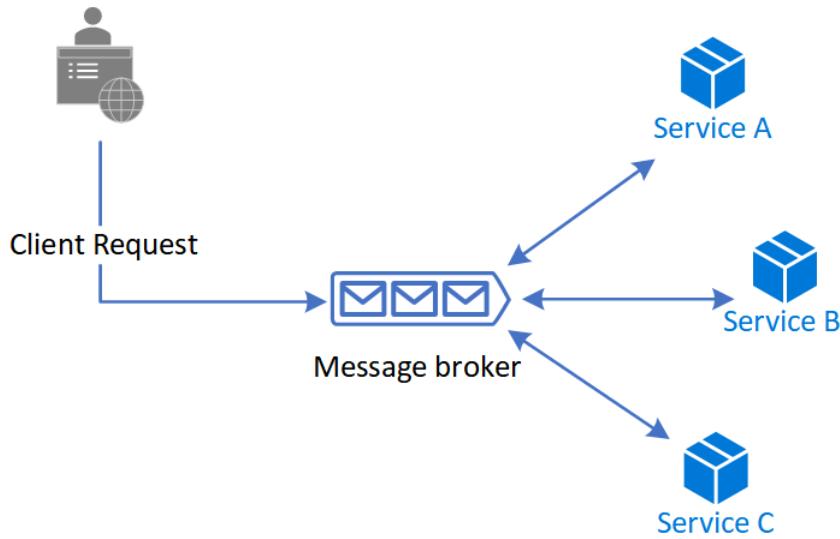
The orchestrator pattern reduces point-to-point communication between services but has some drawbacks because of the tight coupling between the orchestrator and other services that participate in processing of the business transaction. To execute tasks in a sequence, the orchestrator needs to have some domain knowledge about the responsibilities of those services. If you want to add or remove services, existing logic will break, and you'll need to rewire portions of the communication path. While you can configure the workflow, add or remove services easily with a well-designed orchestrator, such an implementation is complex and hard to maintain.



Solution

Let each service decide when and how a business operation is processed, instead of depending on a central orchestrator.

One way to implement choreography is to use the [asynchronous messaging pattern](#) to coordinate the business operations.



A client request publishes messages to a message queue. As messages arrive, they are pushed to subscribers, or services, interested in that message. Each subscribed service does their operation as indicated by the message and responds to the message queue with success or failure of the operation. In case of success, the service can push a message back to the same queue or a different message queue so that another service can continue the workflow if needed. If an operation fails, the message bus can retry that operation.

This way, the services choreograph the workflow among themselves without depending on an orchestrator or having direct communication between them.

Because there isn't point-to-point communication, this pattern helps reduce coupling between services. Also, it can remove the performance bottleneck caused by the orchestrator when it has to deal with all transactions.

When to use this pattern

Use the choreography pattern if you expect to update, remove, or add new services frequently. The entire app can be modified with lesser effort and minimal disruption to existing services.

Consider this pattern if you experience performance bottlenecks in the central orchestrator.

This pattern is a natural model for the serverless architecture where all services can be short lived, or event driven. Services can spin up because of an event, do their task, and are removed when the task is finished.

Issues and considerations

Decentralizing the orchestrator can cause issues while managing the workflow.

If a service fails to complete a business operation, it can be difficult to recover from that failure. One way is to have the service indicate failure by firing an event. Another service subscribes to those failed events takes necessary actions such as applying [compensating transactions](#) to undo successful operations in a request. The failed service might also fail to fire an event for the failure. In that case, consider using a retry and, or time out mechanism to recognize that operation as a failure. For an example, see the Example section.

It's simple to implement a workflow when you want to process independent business operations in parallel. You can use a single message bus. However, the workflow can become complicated when choreography needs to occur in a sequence. For instance, Service C can start its operation only after Service A and Service B have completed their operations with success. One approach is to have multiple message buses that get messages in the required order. For more information, see the [Example](#) section.

The choreography pattern becomes a challenge if the number of services grow rapidly. Given the high number of independent moving parts, the workflow between services tends to get complex. Also, distributed tracing

becomes difficult.

The orchestrator centrally manages the resiliency of the workflow and it can become a single point of failure. On the other hand, for choreography, the role is distributed between all services and resiliency becomes less robust.

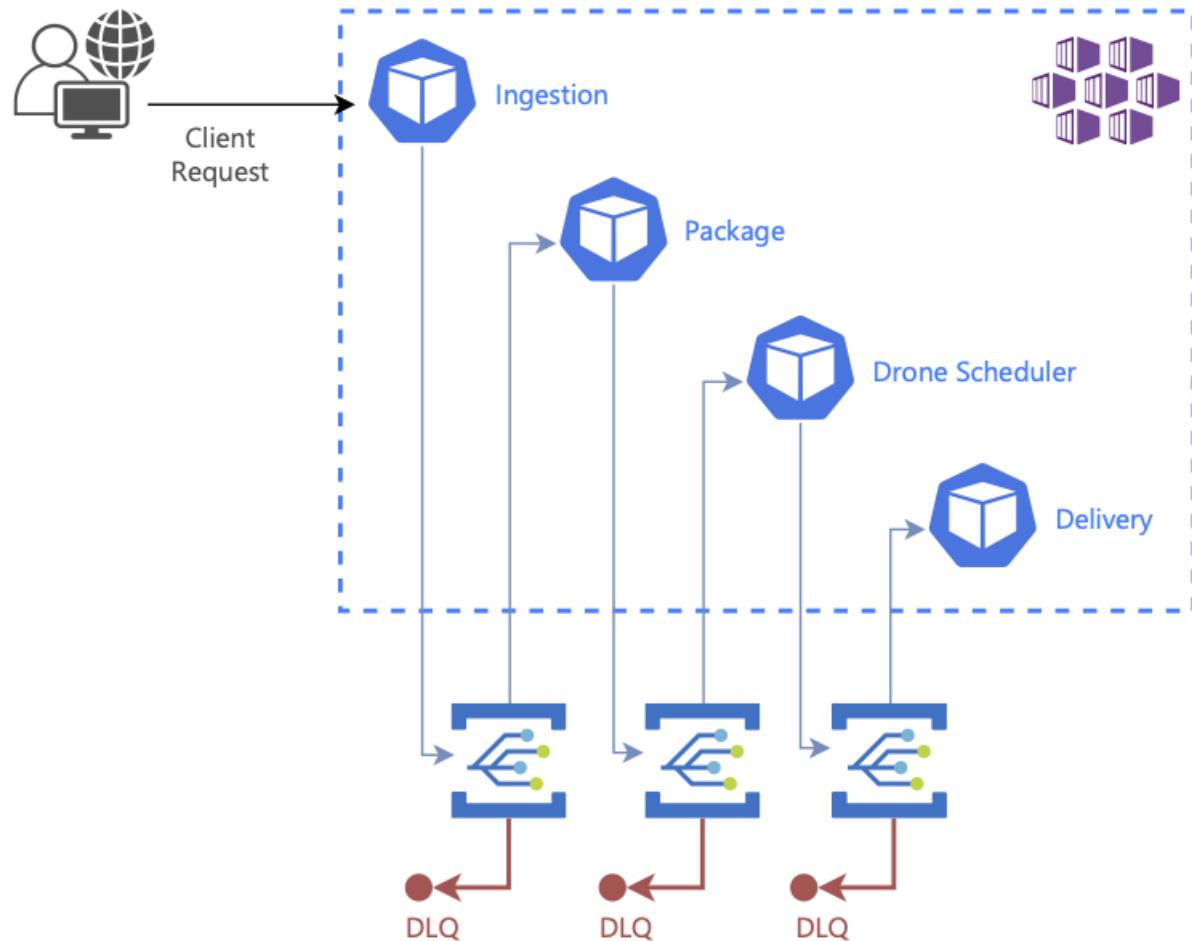
Each service isn't only responsible for the resiliency of its operation but also the workflow. This responsibility can be burdensome for the service and hard to implement. Each service must retry transient, nontransient, and time-out failures, so that the request terminates gracefully, if needed. Also, the service must be diligent about communicating the success or failure of the operation so that other services can act accordingly.

Example

This example shows the choreography pattern with the [Drone Delivery app](#). When a client requests a pickup, the app assigns a drone and notifies the client.



An example of this pattern is available on [GitHub](#).



A single client business transaction requires three distinct business operations: creating or updating a package, assigning a drone to deliver the package, and checking the delivery status. Those operations are performed by three microservices: Package, Drone Scheduler, and Delivery services. Instead of a central orchestrator, the services use messaging to collaborate and coordinate the request among themselves.

Design

The business transaction is processed in a sequence through multiple hops. Each hop has a message bus and the respective business service.

When a client sends a delivery request through an HTTP endpoint, the Ingestion service receives it, raises an operation event, and sends it to a message bus. The bus invokes the subscribed business service and sends the

event in a POST request. On receiving the event, the business service can complete the operation with success, failure, or the request can time out. If successful, the service responds to the bus with the Ok status code, raises a new operation event, and sends it to the message bus of the next hop. In case of a failure or time-out, the service reports failure by sending the BadRequest code to the message bus that sent the original POST request. The message bus retries the operation based on a retry policy. After that period expires, message bus flags the failed operation and further processing of the entire request stops.

This workflow continues until the entire request has been processed.

The design uses multiple message buses to process the entire business transaction. [Microsoft Azure Event Grid](#) provides the messaging service. The app is deployed in an [Azure Kubernetes Service \(AKS\)](#) cluster with [two containers in the same pod](#). One container runs the [ambassador](#) that interacts with Event Grid while the other runs a business service. The approach with two containers in the same pod improves performance and scalability. The ambassador and the business service share the same network allowing for low latency and high throughput.

To avoid cascading retry operations that might lead to multiple efforts, only Event Grid retries an operation instead of the business service. It flags a failed request by sending a messaging to a [dead letter queue \(DLQ\)](#).

The business services are idempotent to make sure retry operations don't result in duplicate resources. For example, the Package service uses upsert operations to add data to the data store.

The example implements a custom solution to correlate calls across all services and Event Grid hops.

Here's a code example that shows the choreography pattern between all business services. It shows the workflow of the Drone Delivery app transactions. Code for exception handling and logging have been removed for brevity.

```

[HttpPost]
[Route("/api/[controller]/operation")]
[ProducesResponseType(typeof(void), 200)]
[ProducesResponseType(typeof(void), 400)]
[ProducesResponseType(typeof(void), 500)]

public async Task<IActionResult> Post([FromBody] EventGridEvent[] events)
{

    if (events == null)
    {
        return BadRequest("No Event for Choreography");
    }

    foreach(var e in events)
    {

        List<EventGridEvent> listEvents = new List<EventGridEvent>();
        e.Topic = eventRepository.GetTopic();
        e.EventTime = DateTime.Now;
        switch (e.EventType)
        {
            case Operations.ChoreographyOperation.ScheduleDelivery:
            {
                var packageGen = await
packageServiceCaller.UpsertPackageAsync(delivery.PackageInfo).ConfigureAwait(false);
                if (packageGen is null)
                {
                    //BadRequest allows the event to be reprocessed by Event Grid
                    return BadRequest("Package creation failed.");
                }

                //we set the event type to the next choreography step
                e.EventType = Operations.ChoreographyOperation.CreatePackage;
                listEvents.Add(e);
                await eventRepository.SendEventAsync(listEvents);
                return Ok("Created Package Completed");
            }
            case Operations.ChoreographyOperation.CreatePackage:
            {
                var droneId = await
droneSchedulerServiceCaller.GetDroneIdAsync(delivery).ConfigureAwait(false);
                if (droneId is null)
                {
                    //BadRequest allows the event to be reprocessed by Event Grid
                    return BadRequest("could not get a drone id");
                }

                e.Subject = droneId;
                e.EventType = Operations.ChoreographyOperation.GetDrone;
                listEvents.Add(e);
                await eventRepository.SendEventAsync(listEvents);
                return Ok("Drone Completed");
            }
            case Operations.ChoreographyOperation.GetDrone:
            {
                var deliverySchedule = await deliveryServiceCaller.ScheduleDeliveryAsync(delivery,
e.Subject);
                return Ok("Delivery Completed");
            }
            return BadRequest();
        }
    }
}

```

Related guidance

Consider these patterns in your design for choreography.

- Modularize the business service by using the [ambassador design pattern](#).
- Implement [queue-based load leveling pattern](#) to handle spikes of the workload.
- Use asynchronous distributed messaging through the [publisher-subscriber pattern](#).
- Use [compensating transactions](#) to undo a series of successful operations in case one or more related operation fails.
- For information about using a message broker in a messaging infrastructure, see [Asynchronous messaging options in Azure](#).

Edge Workload Configuration pattern

3/10/2022 • 6 minutes to read • [Edit Online](#)

The great variety of systems and devices on the shop floor can make workload configuration a difficult problem. This article provides approaches to solving it.

Context and problem

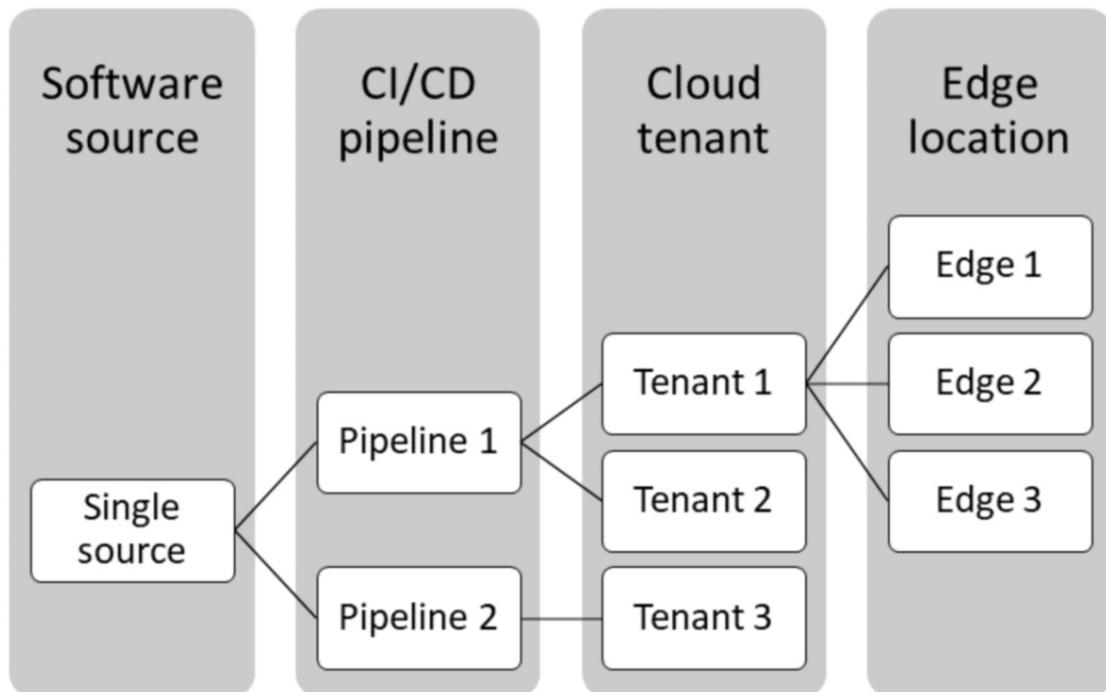
Manufacturing companies, as part of their digital transformation journey, focus increasingly on building software solutions that can be reused as shared capabilities. Due to the variety of devices and systems on the shop floor, the modular workloads are configured to support different protocols, drivers, and data formats. Sometimes even multiple instances of a workload are run with different configurations in the same edge location. For some workloads, the configurations are updated more than once a day. Therefore, configuration management is increasingly important to the scaling out of edge solutions,

Solution

There are a few common characteristics of configuration management for edge workloads:

- There are several configuration points that can be grouped into distinct layers, like software source, CI/CD pipeline, cloud tenant, and edge location:

Layers where configuration can be applied for edge workloads



- The various layers can be updated by different people.
- No matter how the configurations are updated, they need to be carefully tracked and audited.
- For business continuity, it's required that configurations can be accessed offline at the edge.
- It's also required that there's a global view of configurations that's available on the cloud.

Issues and considerations

Consider the following points when deciding how to implement this pattern:

- Allowing edits when the edge isn't connected to the cloud significantly increases the complexity of configuration management. It's possible to replicate changes to the cloud, but there are challenges with:
 - User authentication, because it relies on a cloud service like Azure Active Directory.
 - Conflict resolution after reconnection, if workloads receive unexpected configurations that require manual intervention.
- The edge environment can have network-related constraints if the topology complies to the ISA-95 requirements. You can overcome such restraints by selecting a technology that offers connectivity across layers, such as [device hierarchies](#) in [Azure IoT Edge](#).
- If run-time configuration is decoupled from software releases, configuration changes need to be handled separately. To offer history and rollback features, you need to store past configurations in a datastore in the cloud.
- A fault in a configuration, like a connectivity component configured to a nonexistent end-point, can break the workload. Therefore, it's important to treat configuration changes as you treat other deployment lifecycle events in the observability solution, so that the observability dashboards can help correlate system errors to configuration changes. For more information about observability, see [Cloud monitoring guide: Observability](#).
- Understand the roles that the cloud and edge datastores play in business continuity. If the cloud datastore is the single source of truth, then the edge workloads should be able to restore intended states by using automated processes.
- For resiliency, the edge datastore should act as an offline cache. This takes precedence over latency considerations.

When to use this pattern

Use this pattern when:

- There's a requirement to configure workloads outside of the software release cycle.
- Different people need to be able to read and update configurations.
- Configurations need to be available even if there's no connection to the cloud.

Example workloads:

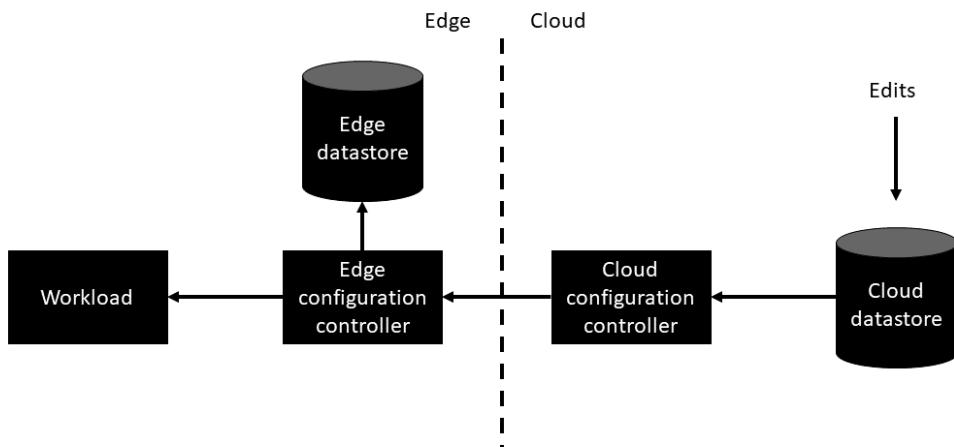
- Solutions that connect to assets on the shop floor for data ingestion—OPC Publisher, for example—and command and control
- Machine learning workloads for predictive maintenance
- Machine learning workloads that inspect in real-time for quality on the manufacturing line

Examples

The solution to configure edge workloads during run-time can be based on an external configuration controller or an internal configuration provider.

External configuration controller variation

External configuration controller



This variation has a configuration controller that's external to the workload. The role of the cloud configuration controller component is to push edits from the cloud datastore to the workload through the edge configuration controller. The edge also contains a datastore so that the system functions even when disconnected from the cloud.

With IoT Edge, the edge configuration controller can be implemented as a module, and the configurations can be applied with [module twins](#). The module twin has a size limit; if the configuration exceeds the limit, the solution can be [extended with Azure Blob Storage](#) or by chunking larger payloads over [direct methods](#).

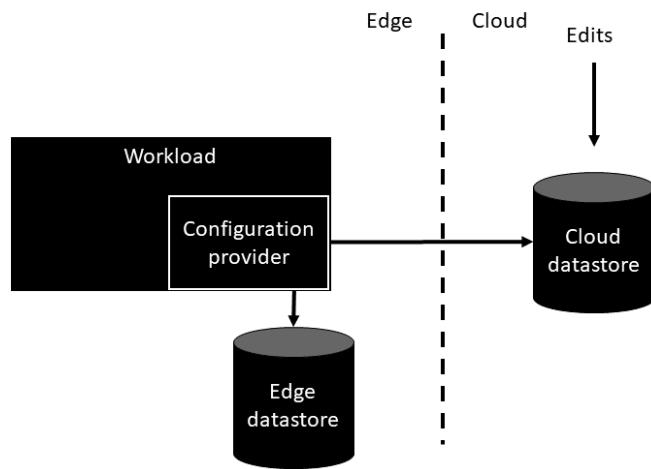
For an end-to-end example of the external configuration controller variation, see the [Connected factory signal pipeline](#).

The benefits of this variation are:

- The workload itself doesn't have to be aware of the configuration system. This capability is a requirement if the source code of the workload is not editable—for example, when using a module from the [Azure IoT Edge Marketplace](#).
- It's possible to change the configuration of multiple workloads at the same time by coordinating the changes via the cloud configuration controller.
- Additional validation can be implemented as part of the push pipeline—for example, to validate existence of endpoints at the edge before pushing the configuration to the workload.

Internal configuration provider variation

Internal configuration provider



In the internal configuration provider variation, the workload pulls configurations from a configuration provider. For an implementation example, see [Implement a custom configuration provider in .NET](#). That example uses C#, but other languages can be used.

In this variation, the workloads have unique identifiers so that the same source code running in different environments can have different configurations. One way to construct an identifier is to concatenate the hierarchical relationship of the workload to a top-level grouping such as a tenant. For IoT Edge, it could be a combination of Azure resource group, IoT hub name, IoT Edge device name, and module identifier. These values together form a unique identifier that work as a key in the datastores.

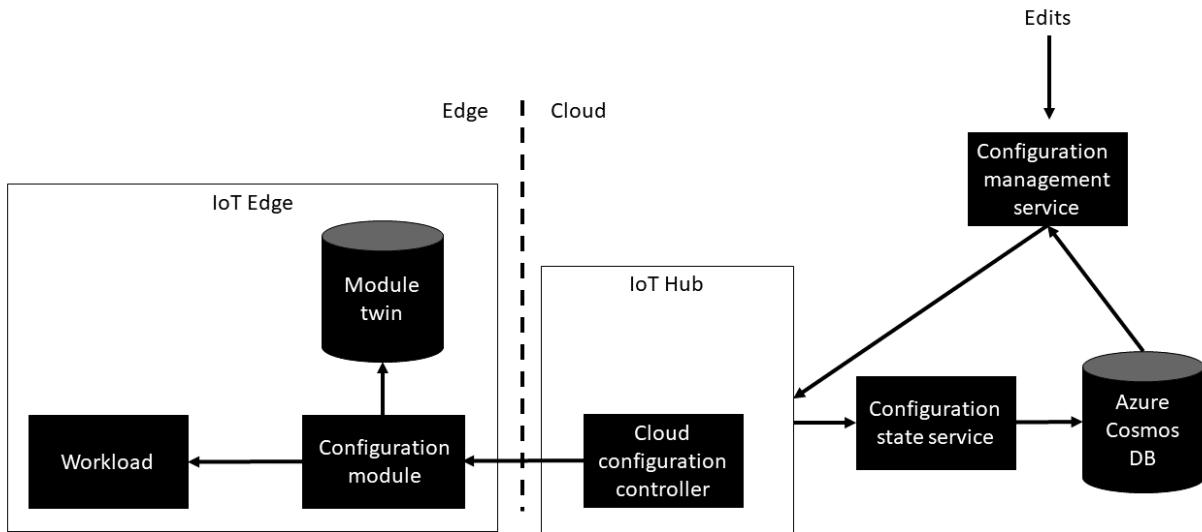
Although the module version can be added to the unique identifier, it's a common requirement to persist configurations across software updates. If the version is a part of the identifier, the old version of the configuration should be migrated forward with an additional implementation.

The benefits of this variation are:

- Other than the datastores, the solution doesn't require components, reducing complexity.
- Migration logic from incompatible old versions can be handled within the workload implementation.

Solutions based on IoT Edge

Solution architecture based on IoT Edge



The cloud component of the IoT Edge reference implementation consists of an IoT hub acting as the cloud configuration controller. The [Azure IoT Hub](#) module twin functionality propagates configuration changes and information about the currently applied configuration by using module twin desired and reported properties. The configuration management service acts as the source of the configurations. It can also be a user interface for managing configurations, a build system, and other tools used to author workload configurations.

An [Azure Cosmos DB](#) database stores all configurations. It can interact with multiple IoT hubs, and provides configuration history.

After an edge device indicates via reported properties that a configuration was applied, the configuration state service notes the event in the database instance.

When a new configuration is created in the configuration management service, it is stored in Azure Cosmos DB and the desired properties of the edge module are changed in the IoT hub where the device resides. The configuration is then transmitted by IoT Hub to the edge device. The edge module is expected to apply the configuration and report via the module twin the state of the configuration. The configuration state service then listens to twin change events, and upon detecting that a module reports a configuration state change, records the corresponding change in the Azure Cosmos DB database.

The edge component can use either the external configuration controller or internal configuration provider. In either implementation, the configuration is either transmitted in the module twin desired properties, or in case large configurations need to be transmitted, the module twin desired properties contain a URL to [Azure Blob Storage](#) or to another service that can be used to retrieve the configuration. The module then signals in the module twin reported properties whether the new configuration was applied successfully and what configuration is currently applied.

Next steps

- [Azure IoT Edge](#)
- [What is Azure IoT Edge?](#)
- [Azure IoT Hub](#)
- [IoT Concepts and Azure IoT Hub](#)
- [Azure Cosmos DB](#)
- [Welcome to Azure Cosmos DB](#)
- [Azure Blob Storage](#)

- [Introduction to Azure Blob storage](#)

Related guidance

- [External Configuration Store pattern](#)

Solutions for the retail industry

3/10/2022 • 4 minutes to read • [Edit Online](#)

Retail is one of the fastest growing industries worldwide, generating some of the biggest revenues and accounting to almost a third of American jobs. The core of retail industry is selling products and services to consumers, through channels such as, storefront, catalog, television, and online. Retailers can enhance or reimagine their customer's journey using Microsoft Azure services by:

- keeping their supply chains agile and efficient,
- unlocking new opportunities with data and analytics,
- creating innovative customer experiences using mixed reality, AI, and IoT, and
- building a personalized and secure multi-channel retail experience for customers.

Using Azure services, retailers can easily achieve these goals. For use cases and customer stories, visit [Azure for retail](#). Microsoft is also revolutionizing the retail industry, by providing a comprehensive retail package, [Microsoft Cloud for Retail](#).

NOTE

Learn more about a retail company's journey to cloud adoption, in [Cloud adoption for the retail industry](#).

Architecture guides for retail

The following articles provide more details about retail architectural topics. Although they are mostly conceptual, they can also include implementation details.

GUIDE	SUMMARY	TECHNOLOGY FOCUS
Data Management in Retail	Primer for how to ingest, prepare, store, analyze, and take action on data, for the retail industry.	Databases
Migrate your e-commerce solution to Azure	Learn how to move an existing e-commerce solution to the cloud. The three stages are to rehost, refactor, and rebuild your solution.	Migration
Optimize and reuse an existing recommendation system	The process of successfully reusing and improving an existing recommendation system that is written in R.	AI/ML
Visual search in retail with CosmosDB	This document focuses on the AI concept of visual search and offers a few key considerations on its implementation. It provides a workflow example and maps its stages to the relevant Azure technologies.	Databases

GUIDE	SUMMARY	TECHNOLOGY FOCUS
SKU optimization for consumer brands	Topics include automating decision making, SKU assortment optimization, descriptive analytics, predictive analytics, parametric models, non-parametric models, implementation details, data output and reporting, and security considerations.	Analytics

Architectures for retail

The following articles provide detailed analysis of architectures developed and recommended for the retail industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Batch scoring with R models to forecast sales	Perform batch scoring with R models using Azure Batch. Azure Batch works well with intrinsically parallel workloads and includes job scheduling and compute management.	IoT
Batch scoring with R models to forecast sales	Perform batch scoring with R models using Azure Batch. Azure Batch works well with intrinsically parallel workloads and includes job scheduling and compute management.	AI/ML
Build a content-based recommendation system	This example scenario shows how your business can use machine learning to automate content-based personalization for your customers.	AI/ML
Build a Real-time Recommendation API on Azure	Build a recommendation engine that can be generalized for products, movies, news, and other consumer services, using Azure Databricks, Azure Machine Learning, Azure Cosmos DB, and Azure Kubernetes Service.	AI/ML
Data warehousing and analytics	Build an insightful sales and marketing solution with a data pipeline that integrates large amounts of data from multiple sources into a unified analytics platform in Azure.	Analytics
E-commerce front end	Implement a scalable and cost-effective e-commerce front end using Azure platform as a service (PaaS) tools.	Web
IBM z/OS online transaction processing on Azure	With a dynamically adaptable infrastructure, businesses can realize and launch their products quickly to delight their users. Learn how to migrate a z/OS mainframe OLTP application to a secure, scalable, and highly available system in the cloud.	Mainframe

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Intelligent product search engine for e-commerce	Use Azure Cognitive Search, a dedicated search service, to dramatically increase the relevance of search results for your e-commerce customers.	Web
Magento e-commerce platform in Azure Kubernetes Service	Learn how to deploy and host Magento, an open-source e-commerce platform, on Azure.	Web
Movie recommendations on Azure	Automate movie and product recommendations by using an Azure Data Science Virtual Machine to train an Azure Machine Learning model.	AI/ML
Retail - Buy online, pickup in store (BOPIS)	Develop an efficient and secure curbside pickup process on Azure.	Web
Scalable order processing	Build a highly scalable and resilient architecture for online order processing, using managed Azure services, such as Cosmos DB and HDInsight.	Web
Stream processing with Azure Databricks	Use Azure Databricks to build an end-to-end stream processing pipeline for a taxi company, to collect, and analyze trip and fare data from multiple devices.	Analytics
Stream processing with Azure Stream Analytics	Use Azure Stream Analytics to build an end-to-end stream processing pipeline for a taxi company, to collect, and analyze trip and fare data from multiple devices.	Analytics

Solution ideas for retail

The following are other ideas that you can use as a starting point for your retail solution.

AI:

- [Commerce Chatbot with Azure Bot Service](#)
- [Customer Feedback and Analytics](#)
- [FAQ Chatbot with data champion model](#)
- [Interactive Voice Response Bot](#)
- [Optimize Marketing with Machine Learning](#)
- [Personalized marketing solutions](#)
- [Personalized Offers](#)
- [Predictive Marketing with Machine Learning](#)
- [Product recommendations for retail](#)
- [Retail Assistant with Visual Capabilities](#)

Analytics:

- Big data analytics with Azure Data Explorer
- Demand forecasting and price optimization
- Demand forecasting with Azure Machine Learning
- Demand forecasting for shipping and distribution
- Interactive price analytics
- Modern analytics architecture with Azure Databricks

Databases:

- Retail and e-commerce using Azure MySQL
- Retail and e-commerce using Azure PostgreSQL
- Retail and e-commerce using Cosmos DB

Mixed Reality:

- Facilities management powered by mixed reality and IoT

Networking:

- Video capture and analytics for retail

Web:

- E-commerce website running in secured App Service Environment
- Architect a scalable e-commerce web app
- Scalable Episerver marketing website
- Scalable Sitecore marketing website
- Simple digital marketing website

Data management in the retail industry

3/10/2022 • 11 minutes to read • [Edit Online](#)

Introduction

Data is the foundation for developing and delivering better retail experiences. Data is found in every facet of a retail organization and can be used to extract insights across the value chain into operational performance and customer behavior, as well as leveraged to power improved service experiences. From online browsing to social engagement to in-store purchasing, data abounds. However, capturing data is only a portion of data management. Stitching together disparate data for analysis requires proper handling of data across an organization—thus improving a retailer's ability to make impactful decisions about running their business.

For example, with the growth of mobile shopping, customers have come to expect that retailers have a reasonable amount of data about their shopping habits to be used to improve the experience. A use case example is a personalized product and promotion offering sent directly to a customer's mobile device when shopping in a specific location within a physical retail store. Leveraging data on what, where, how, how many and how often, plus additional inputs such as store product availability, creates opportunities to send real-time promotion messages to a customer's device when the customer is shopping in proximity of a targeted product.

Effective data usage can activate the customer to buy by helping the retailer delivering a more relevant experience; for example, retailers might send the customer a notification with a discount code for the retailer's eCommerce website. Further, this data will drive actionable insights from which company leaders may steer their actions with data-backed decisions

The action to offer a promotion is informed by a combination of data points and triggered by the customer entering the store. The ability to make these connections and the resulting actions are based on the data management model shown below.



Figure 1

When bringing data into Azure, consider the 3Ps of data sources and their applicability to the scenarios the retailer wants to enable. The 3Ps of data sources are Purchased, Public, and Proprietary.

Purchased data typically augments and enhances the organization's existing data most often with market and demographic data that supplements the organization's data capture reach. For example, a retailer may purchase additional demographic data to augment a master customer record, ensuring the record is accurate and complete.

Public data is freely available and may be harvested from social media, government resources (e.g. geography), and other online sources. This data can infer insights such as weather patterns that correlate with purchasing patterns or social engagement that signals product popularity amongst a specific geography. Public data is often available via APIs.

Proprietary data resides within the organization. It may be a retailer's on-premises systems, SaaS applications, or cloud providers. To access the data in a SaaS application provider, and other vendor data, APIs are typically used to communicate the vendor's system. This includes data such as eCommerce site logs, POS sales data, and inventory management systems.

These different data types are used for various insights coming from the data management pipeline.

Ingest

Initially, data is loaded into Azure in its native format, and is stored accordingly. Receiving and managing disparate data sources can be daunting, but Microsoft Azure offers services to load data into the cloud quickly and easily, making it available for processing in the data management pipeline.

Azure has several helpful services for migrating data. The choice depends on the type of data being migrated. [Azure Data Migration Services](#) for SQL Server and the [Azure Import/Export Service](#) are services to help get data into Azure. Other data ingress services to consider include [Azure Data Factory](#) and [Azure Logic Apps](#) connectors. Each has its own features and should be investigated to see which technology works best for the given situation.

Data ingestion isn't limited to Microsoft technologies. Through the [Azure Marketplace](#), retailers may configure many different vendor databases in Azure to work with existing on-premises systems.

Not all data must be maintained in Azure. For example, Point of Sale (POS) data may be held on-premises so Internet outages do not impact sales transactions. This data can be queued and uploaded to Azure on a schedule (perhaps nightly or weekly) for use in analysis, but always treating the on-premises data as the source of truth.

Prepare

Before analysis begins, the data must be prepared. This shaping of data is important to ensure quality of predictive models, reporting KPIs and relevancy of data.

There are two types of data to address when preparing data for analysis, structured and unstructured. Structured data is easier to deal with since it is already formed and formatted. It may require just a simple transformation to go from structured data in source format to structured data which is ready for analysis jobs. Unstructured data typically provides more challenges. Unstructured data isn't stored in a fixed record length format. Examples include documents, social media feeds, and digital images and videos. These data must be managed differently than structured data and often require a dedicated process to ensure these data end up in the right data store, in a useable way.

Data shaping occurs during the Extract-Transform-Load (ETL) process, in the preparation stage. Data is extracted from the unchanged data sources imported into Azure, "cleaned" or reformatted as needed, and stored in a new, more structured format. A common ETL data preparation operation is to transform .csv or Excel files into parquet files, which are easier for machine learning systems like Apache Spark to read and process quickly. Another common scenario is to create XML files or JSON from .csv files, or other formats. The resulting format is easier to use with other analysis engines.

In Azure, there are several transformation technologies available as a ETL services to reshape data. Options include [Azure Databricks](#), [Azure Functions](#) or Logic Apps. Databricks is a fully managed instance of Apache Spark, and is used to transform data from one form to another. Azure Functions are stateless (or "serverless") functions with triggers to fire them and run code. Logic Apps integrates services.

Store

Storing data before processing requires consideration. Data can come in structured or unstructured formats and the shape of the data often determines its storage destination. For example, highly structured data may be suitable for Azure SQL. Less structured data may be held in blob storage, file storage, or table storage.

Data stored in Azure has great performance backed up by a solid service-level agreement (SLA). Data services provide easier to manage solutions, high availability, replication across multiple geographic locations and—above all—Azure offers the data stores and services needed to drive Machine Learning.

Both structured and unstructured data can be stored in [Azure Data Lake](#) and queried using [U-SQL](#), a query

language specific to Azure Data Lake. Examples of data that may be included in a Data Lake include the following, which are divided into commonly structured and unstructured data sources.

Structured data

- CRM data and other line of business applications
- POS transaction data
- Sensor data
- Relational data
- eCommerce transaction data

Unstructured data

- Social feeds
- Video
- Digital images
- Website clickstream analysis

There are a growing number of use cases supporting unstructured data to generate value. This is propelled by the desire for data-driven decisions and the advancement in technology such as AI to enable capture and processing of data at scale. For example, data can include photos or streaming video. For example, streaming video can be leveraged to detect customer shopping selections for a seamless checkout; or product catalog data can be merged seamlessly with a customer's photo of their favorite dress to provide a view of similar, or recommended items.

Examples of structured data include relational database data feeds, sensor data, Apache Parquet files, and ecommerce data. The inherent structure of these data makes them well-suited for a Machine Learning pipeline.

Azure Data Lake service also enables batch and interactive queries along with real time analytics using [Data Lake Analytics](#). Also, Data Lake is specifically well-suited for very large data analysis workloads. Finally, data in the Data Lake is persistent and has no time limit.

Other data stores such as relational databases, Blob storage, Azure Files storage, and Cosmos DB document storage may also hold clean data ready for downstream analysis in the data management pipeline. There is no requirement that one uses a Data Lake.

Analyze

For problems like reducing cost of inventory, retailors can use analysis performed by a Machine Learning process.

Data analysis prepares data for processing through a Machine Learning engine to gain deeper insights into the customer experience. This process produces a model that "learns" and may be applied to future data to predict outcomes. Models define the data that will be examined and how the data will be analyzed through various algorithms. Using the output data from the analysis with data visualization is what could trigger an insight—such as offering an in-store coupon for an item from the customer's wish list in the retailors eCommerce platform.

Data analysis occurs by feeding learning ecosystems with data stored for processing. Typically, this is machine learning performed by Hadoop, Databricks, or a self-managed Spark instance running on a virtual machine. This can also be done simply by querying for data. Insight into KPIs can often be found in clean data without going through a machine learning pipeline.

[Hadoop](#) is part of the fully managed Azure service, [HDInsight](#). HDInsight is a collection of data learning tools used for training data models, outputting data to a data warehouse, and performing queries on Hadoop through the Hive query language. HDInsight can analyze streaming or historical data.

A variety of learning algorithms may be applied to the data as part of training and to maintain data models. A data model explicitly determines the structure of data produced for analysts.

First, the data is cleaned and formed appropriately. It is then processed by a machine learning system such as HDInsight or Apache Spark. To do this, existing data is used to train a model, which in turn is used in analysis of data. The trained model is updated periodically with new known good data to increase its accuracy during analysis. Machine learning services use the model to perform an analysis of the data being processed.

After model training and running a data analysis process, data derived from machine learning analysis can be stored in a data warehouse, or normalized storage databases for analytics data. Microsoft provides [Power BI](#), a fully featured data analytics tool, for deep analysis of data in the data warehouse.

Action

Data in retail moves constantly, and systems that handle it must do so in a timely manner. For example, eCommerce shopper data needs to be processed quickly. This is so items in a buyer's cart can be used to offer additional services, or add-on items during the checkout process. This form of data handling and analysis must occur almost immediately and is typically carried out by systems performing "micro-batch" transactions. That is, data is analyzed in a system which has access to already processed data and is run through a model.

Other "batch" operations may occur at regular intervals but need not occur in near real time. When batch analysis occurs on-premises, these jobs often run at night, on weekends, or when resources are not in use. With Azure, scaling large batch jobs and the virtual machines needed to support them may occur at any time.

Use the following steps to get started.

1. Create a data ingestion plan for data stores providing value to the analysis to be performed. With a detailed data synchronization or migration plan in place, get the data into Azure in its original format.
2. Determine the actionable insights needed and choose a data processing pipeline to accommodate the data processing activities.
3. With these data features in mind, create a data processing pipeline using the appropriate algorithms to gain the insights being sought.
4. Use a common data model for output into a data warehouse, if possible; this can expose the most interesting data features. This usually means reading data in the original Azure storage systems and writing the cleaned version to another data store.
5. Process the data through the machine learning pipelines provided by Spark or Hadoop. Then feed the output to a data warehouse. There are many default algorithms to process the data, or retailers can implement their own. In addition to ML scenarios, load data into standard data storage and enforce a common data model, then query for KPI data. For example, data may be stored in a star schema or other data store.

With data now ready to be used by data analysts, actionable insights may be discovered, and action taken to exploit this new knowledge. For example, a customer's purchase preferences may be loaded back into the retailer's systems and used to improve several customer touchpoints such as the following.

- Increase the average eCommerce or POS transaction by bundling products
- Purchase history in CRM to support customer call center inquiries
- Product suggestions tailored by an e-commerce recommendation engine
- Targeted and relevant ads based on customer data
- Updated inventory availability based on product movement within the supply chain

Another type of insight that may arise are patterns not previously questioned. For example, it may be discovered that more inventory loss happens between the hours of 3:00 PM and 7:00 PM. This might imply the need for

additional data to determine a root cause and a course of action—such as improved security or standard operating procedures.

Conclusion

Data management in retail is complex. But it offers the valuable ability to deliver relevance and an improved customer experience. Using the techniques in this article, insights may be gained to improve the customer experience, drive profitable business outcomes and uncover trends that may drive operational improvements.

Contributors

This article is being updated and maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [David Starr](#) | Principal Solutions Architect
- [Mariya Zorotovich](#) | Head of Customer Experience, HLS & Emerging Technology

Next steps

To continue to understand more of Azure capabilities related to implementing a data management pipeline, read the following:

- See how [Azure Data Factory](#) can help ingest data from on-premises data stores into Azure.
- Learn more about how [Azure Data Lake](#) can serve as a store all data, both structured and unstructured.
- See actual retail reports illustrating how [Power BI](#) can give deeper insights into known questions, but enable trend analysis.
- Visit the [Azure Marketplace](#) to find solutions compatible with those already on-premises.

Migrate your e-commerce solution to Azure

3/10/2022 • 15 minutes to read • [Edit Online](#)

Introduction

Moving an existing e-commerce solution to the cloud presents many benefits for an enterprise: it enables scalability, it offers customers 24/7 accessibility, and it becomes easier to integrate cloud services. But first, to move an e-commerce solution to the cloud is a significant task, with costs that must be understood by a decision maker. This document explains the scope of an Azure migration with the goal of informing you of the options. The first phase begins with IT Pros moving the components to the cloud. Once you're on Azure, we describe the steps an e-commerce team can take to increase your return on investment (ROI) and to take advantage of the cloud.

At the crossroads

Although global e-commerce transactions account for only a fraction of total retail sales, the channel continues to see steady year-over-year growth. In 2017, e-commerce constituted 10.2% of total retail sales, up from 8.6% in 2016 ([source](#)). As e-commerce has matured, along with the advent of cloud computing, retailers are at a crossroads. There are choices to make. They can envision their business model with new capabilities made possible by evolving technology; and they can plan their modernization given their current capability footprint.

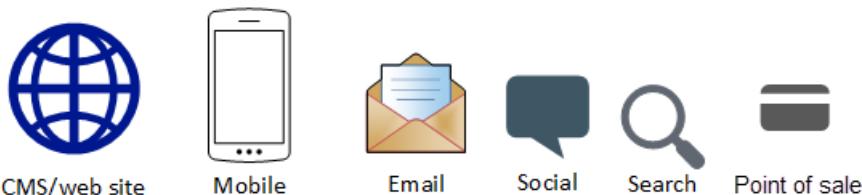
Improving the customer journey

E-commerce, which is primarily focused on the customer journey, has many different attributes. These attributes can be grouped into four main areas: discovery, evaluation, purchase and post-purchase.

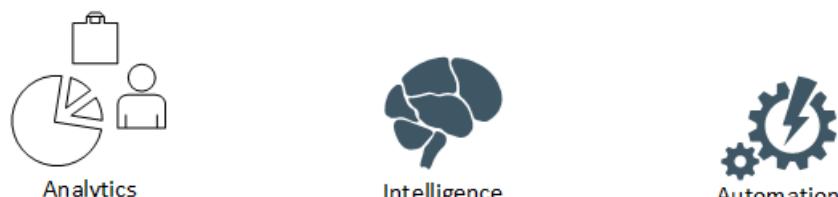
The customer behavior is captured as data. The shopping funnel is a collection of connection points to applications used for viewing product data, transactions, inventory, shipping, order fulfillment, customer profile, shopping cart, and product recommendations, to name a few.

A typical retail business relies on a large collection of software solutions that range from customer-facing applications, down through the stack to foundational applications. The following drawing shows a view of the functionality present in a typical retail business.

Externally visible functionality



Core functionality



The cloud presents an opportunity to shift how an organization obtains, uses and manages technology. Other benefits include: reduced costs of maintaining data centers, improved reliability and performance, and the flexibility to add other services. In this use case, we look at a path a retail business can take to migrate its existing infrastructure to Azure. We also take advantage of the new environment using a phased approach of rehost, refactor and rebuild. While many organizations may follow this series path to modernization, in most cases, organizations can drop into any phase as their starting point. Organizations may choose to forgo rehosting their current application on Azure, and jump straight to refactor or even rebuild. This decision will be unique to the application, and organization to best meet their modernization needs.

Rehost

Also referred to as "lift and shift," this stage entails migrating physical servers and VMs as-is to the cloud. By simply shifting your current server environment straight to IaaS, you reap the benefits of cost savings, security, and increased reliability. The savings come from techniques like running workloads on properly sized VMs. Today, the capabilities of on-premises VMs and physical machines frequently exceed the day-to-day needs of retailers. The VMs must be able to handle seasonal business peaks that occur only a few times a year. Therefore you are paying for unused capabilities during the off-peak season. With Azure, you pick the right sized VM based on demands for the current business cycle.

To rehost in Azure, there are three phases:

- **Analysis** : Identify and inventory on-premises resources like applications, workloads, networking, and security. At the end of this phase, you have complete documentation of the existing system.
- **Migration** : Move each subsystem from on-premises to Azure. During this stage, you'll use Azure as an extension of your data center with the applications continuing to communicate.
- **Optimization** : As systems move into Azure, make sure that things are sized properly. If the environment shows that too many resources are allocated to some VMs, change the VM type to one that has a more appropriate combination of CPU, memory, and local storage.

Analyze

Take the following steps:

1. List the on-premises servers and applications. This process relies on an agent or management tool to gather metadata about the servers, the applications that run on the servers, the current server usage, and how the servers and their applications are configured. The result is a report of all the servers and applications in the environment.
2. Identify the dependencies. You can use tooling to identify which servers talk to each other, and applications that communicate to each other. The result is a map—or maps—of all applications and workloads. These maps feed into migration planning.
3. Analyze the configurations. The goal is to know what VM types you need once running in Azure. The result is a report on all applications that can move to Azure. They can be further classified as having:
 - a. No modifications
 - b. Basic modifications such as naming changes
 - c. Minor modifications, such as a slight code changes
 - d. Incompatible workloads that require extra effort to move
4. Create your budget. You now have a list that enumerates each CPU—memory, and so on—and the requirements for each application. Place those workloads on properly sized VMs. The cloud-platform bill costs are based on usage. Tooling exists to map your needs to the right sized Azure VMs. If you are migrating Windows VMs or SQL Server, you should also look at the [Azure Hybrid Benefit](#), which reduces your expenses on Azure.

Microsoft provides several tools to analyze and catalog your systems. If you run VMware, you can use [Azure Migrate](#) to assist with discovery and assessment. The tool identifies machines that can be moved to Azure,

recommends the type of VM to run, and estimates the cost of the workload. For Hyper-V environments, use [Azure Site Recovery Deployment Planner](#). For large migrations where you need to move hundreds or more VMs, you can work with an [Azure migration partner](#). These partners have the expertise and experience to move your workloads.

Migrate

Begin planning which services to move to the cloud and in what order. Because this stage involves moving workloads, follow this order:

1. Build out the network.
2. Incorporate an identity system (Azure Active Directory).
3. Provision storage pieces in Azure.

During migration, the Azure environment is an extension of your on-premises network. You can connect the logical networks with [Azure Virtual Network](#). You can choose to use [Azure ExpressRoute](#) to keep communications between your network and Azure on a private connection that never touches the Internet. You can also use a site-to-site VPN where an Azure VPN Gateway talks to your on-premises VPN device with all traffic sent securely using encrypted communication between Azure and your network. We have published a reference architecture detailing how to setup a hybrid network [here](#).

Once the network is configured, plan for business continuity. A recommendation is to use real time replication to move your on-premises data to the cloud and to ensure that the cloud and the existing data are the same. Ecommerce stores never close; duplication provides the ability to switch over from on-premises to Azure with minimal impact to your customers.

Begin moving the data, applications, and related servers into Azure. Many companies use the [Azure Site Recovery](#) service to migrate to Azure. The service targets business continuity and disaster recovery (BCDR). This is perfect for a migration from on-premises to Azure. Your implementation team can read the details of how to migrate on-premises VMs and physical servers to Azure [here](#).

Once a subsystem has been moved to Azure, test to make sure that everything works as expected. Once all issues are closed, move the workloads over to Azure.

Optimize

At this point, you'll continue to monitor the environment and change the underlying compute options to fit workloads as the environment changes. Whoever monitors the health of the environment should watch how much each resource is used. The goal should be to have 75-90% utilization on most of the VMs. On VMs that have exceptionally low utilization, consider packing them with more applications, or migrating to the lowest cost VMs on Azure that retain the right level of performance.

Azure provides tools to optimize the environment as well. [Azure Advisor](#) monitors components of your environment and provides personalized recommendations based on best practices. The recommendations help improve the performance, security, and availability of the resources used in your applications. The Azure portal also exposes information about the health of your applications. Your VMs should take advantage of the [Azure virtual machine extensions for Linux and Windows](#). Those extensions provide for post deployment configuration, antivirus, app monitoring, and more. You can also take advantage of many other Azure services for network diagnostics, service usage, and alerting through services like [Network Watcher](#), [Service Map](#), [Application Insights](#), and [Log Analytics](#).

While parts of the organization are optimizing the system now in Azure, the development teams can begin moving to the post-migration phase: refactor.

Refactor

With the migration complete, your ecommerce application can start taking advantage of its new home in Azure.

The refactor phase does not have to wait until the entire environment has moved. If your CMS team has migrated, but the ERP team has not, no problem. The CMS team can still begin their refactoring efforts. This stage involves using additional Azure services to optimize the cost, reliability, and performance by refactoring your applications. Where in lift and shift, you were only taking advantage of the provider managed hardware and OS, in this model you also take advantage of cloud services to drive down cost. You continue to utilize your current application as-is, with some minor application code or configuration changes, and connect your application to new infrastructure services such as containers, database, and identity management systems.

The refactoring effort changes very little code and configuration. You'll focus more time on automation mostly because the technologies adopted at this phase rely upon scripting to build up and deploy the resources; the deployment instructions are a script.

While many of the Azure services can be used, we will focus on the most common services used in the refactor phase: containers, app services, and database services. Why do we look at refactoring? Refactoring provides a strong code foundation that lowers long-term costs by keeping code debt within reason.

Containers provide a way to bundle applications. Because of the way a container virtualizes the operating system, you can pack multiple containers into a single VM. You can move an application to a container with zero to few code changes; you may need configuration changes. This effort also leads to writing scripts that bundle applications into a container. Your development teams will spend their refactoring time writing and testing these scripts. Azure supports containerization through the [Azure Kubernetes Service](#) (AKS) and the related [Azure Container Registry](#) which you can use to manage the container images.

For app services, you can take advantage of various Azure services. For example, your existing infrastructure may handle a customer order by placing messages into a queue like [RabbitMQ](#). (For example, one message is to charge the customer, a second is to ship the order.) When rehosting, you put RabbitMQ in a separate VM. During refactoring, you add a [Service Bus](#) queue or topic to the solution, rewrite your RabbitMQ code, and stop using the VMs that served the queuing functionality. This change replaces a set of VMs with an always-on message queue service for a lower cost. Other app services can be found in the Azure Portal.

For databases, you can move your database from a VM to a service. Azure supports SQL Server workloads with [Azure SQL Database](#) and [Azure SQL Database Managed Instance](#). The [Data Migration Service](#) assesses your database, informs you of work that needs to happen prior to the migration, and then moves the database from your VM to the service. Azure supports [MySQL](#), [PostgreSQL](#), and [other database](#) engine services as well.

Rebuild

Up until this point, we tried to minimize changes to the ecommerce systems—we left working systems alone. Now, let's discuss how to really take advantage of the cloud. This stage means to revise the existing application by aggressively adopting PaaS or even SaaS services and architecture. The process encompasses major revisions to add new functionality or to rearchitect the application for the cloud. *Managed APIs* is a new concept that takes advantage of cloud systems. We can make our system easier to update, by creating APIs for communication between services. A second benefit is the ability to gain insights on the data we have. We do this by moving to a *microservice plus API* architecture and use machine learning and other tools to analyze data.

Microservices + APIs

Microservices communicate through externally facing APIs. Each service is self-contained and should implement a single business capability, for example: recommend items to customers, maintain shopping carts, and so on. Decomposing an application into microservices requires time and planning. While no hard rules exist to define a microservice, the general idea involves reducing the deployable unit to a set of components which almost always change together. Microservices allow you to deploy changes as frequently as needed while reducing the testing burden for the overall application. Some services might be extremely small. For those, going serverless with [Azure Functions](#) works well to scale out to as many callers as needed while consuming no resources when not in use. Other services will be broken out around business capabilities: manage product, capture customer orders, and so on.

Serverless mechanisms do have drawbacks: when under light load, they can be slow to respond as some server in the cloud takes a few seconds to configure and run your code. For parts of your environment used heavily by customers, you want to make sure that they can find products, place orders, request returns, and so on with speed and ease. Any time that performance slows down, you risk losing customers in the shopping funnel. If you have functionality that must respond quickly, rebuild that functionality as individually deployable units in [Azure Kubernetes Service](#). For other cases, such as services which require some combination of lots of memory, several CPUs, and plenty of local storage, it may make sense to host the microservice in its own VM.

Each service uses an API for interaction. Access to the API can be direct to the microservice, but this requires anyone communicating with the service to know the application topology. A service like [API Management](#) gives you a central way to publish APIs. All applications simply connect to the API Management service. Developers can discover what APIs are available. The API Management service also provides capabilities to make your retail environment perform well. The service can limit access to the API by different parts of the application (to prevent bottlenecks), cache responses to slow changing values, convert from JSON to XML, and more. A complete list of policies can be found [here](#).

Make use of your data and the Azure Marketplace

Because you have all your data and systems in Azure, you can easily incorporate other SaaS solutions into your business. You can do some things immediately. For example, use [Power BI](#) to stitch together various data sources to create visualizations and reports—and gain insights.

Next, take a look at the offerings in the [Azure Marketplace](#) which can help you do things like optimize inventory, manage campaigns based on customer attributes, and present the right items to each customer based on their preferences and history. Expect to spend some time configuring your data to work in the Marketplace offerings.

Components

Used during rehost:

- [Azure Advisor](#) is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments.
- The [Azure Migrate](#) service assesses on-premises workloads for migration to Azure.
- [Azure Site Recovery](#) orchestrates and manages disaster recovery for Azure VMs, and on-premises VMs and physical servers.
- [Azure Virtual Network](#) enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks.
- [Azure ExpressRoute](#) lets you extend your on-premises networks into the Microsoft cloud over a private connection facilitated by a connectivity provider.

Used during refactor:

- [Azure Kubernetes Service](#) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise.
- [Azure SQL Database](#) is a general-purpose relational database managed service in Microsoft Azure. It supports structures such as relational data, JSON, spatial, and XML. SQL Database offers managed single SQL databases, managed SQL databases in an elastic pool, and SQL Managed Instances.

Used during rebuild:

- Azure [API Management](#) helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services.
- [Azure Functions](#) is a solution for easily running small pieces of code, or "functions," in the cloud.
- [Power BI](#) is a suite of business analytics tools that deliver insights throughout your organization.

Conclusion

Moving your ecommerce system into Azure takes analysis, planning and a defined approach. We looked at a three phase approach of rehost, refactor, and rebuild. This allows an organization to move from one working state to another while minimizing the amount of change at each step. Retailers may also choose to refactor or even rebuild components, skipping rehosting altogether. Many times, you'll have a clear path forward to modernization—take it when you can. As you gain experience running in Azure, you'll see more opportunities to add new capabilities, reduce costs, and improve the overall system.

Contributors

This article is being updated and maintained by Microsoft. It was originally written by the following contributors:

- [Scott Seely](#) | Software Architect
- [Mariya Zorotovich](#) | Head of Customer Experience, HLS & Emerging Technology

Next steps

Many development teams are tempted to do rehost and refactor simultaneously to address technical debt and better leverage capacity. There are benefits to rehosting before jumping into the next steps. Any issues in the deployment to the new environment will be easier to diagnose and fix. This in turn gives your development and support teams time to ramp up with Azure as the new environment. As you begin to refactor and rebuild the system, you are building on a stable, working application. This allows for smaller, targeted changes and more frequent updates.

We have published a more general whitepaper on migrating to the cloud: [Cloud Migration Essentials](#). This is a great piece to read through as you plan out your migration.

Solutions for the finance industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

The finance industry includes a broad spectrum of entities such as banks, investment companies, insurance companies, and real estate firms, engaged in the funding and money management for individuals, businesses, and governments. Besides data security concerns, financial institutions face unique issues such as, heavy reliance on traditional mainframe systems, cyber and technology risks, compliance issues, increasing competition, and customer expectations. By modernizing and digitally transforming financial systems to move to cloud platforms such as Microsoft Azure, financial institutes can mitigate these issues and provide more value to their customers.

With digital transformation, financial institutions can leverage the speed and security of the cloud and use its capabilities to offer differentiated customer experiences, manage risks, and fight fraud. To learn more, visit [Azure for financial services](#). Banking and capital market institutions can drive innovative cloud solutions with Azure; learn from relevant use cases and documentation at [Azure for banking and capital markets](#). Microsoft also provides a complete set of capabilities across various platforms in the form of [Microsoft Cloud for Financial Services](#).

Architectures for finance

The following articles provide detailed analysis of architectures recommended for the finance industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Decentralized trust between banks	Learn how to establish a trusted environment for information sharing without resorting to a centralized database, in banks or other financial institutions.	Blockchain
Replicate and sync mainframe data in Azure	Replicate and sync mainframe data to Azure for digital transformation of traditional banking systems.	Mainframe
Modernize mainframe & midrange data	End to end modernization plan for mainframe and midrange data sources.	Mainframe
Refactor IBM z/OS mainframe Coupling Facility (CF) to Azure	Learn how to leverage Azure services for scale-out performance and high availability, comparable to IBM z/OS mainframe systems with Coupling Facilities (CFs).	Mainframe
Banking system cloud transformation on Azure	Learn how a major bank modernized its financial transaction system while keeping compatibility with its existing payment system.	Migration

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Patterns and implementations in banking cloud transformation	Learn the design patterns and implementations used for the Banking system cloud transformation on Azure .	Migration
JMeter implementation reference for load testing pipeline solution	Learn about an implementation for a scalable cloud load testing pipeline used for the Banking system cloud transformation on Azure .	Migration
Real-time fraud detection	Learn how to analyze data in real time to detect fraudulent transactions or other anomalous activity.	Security

Solution ideas for finance

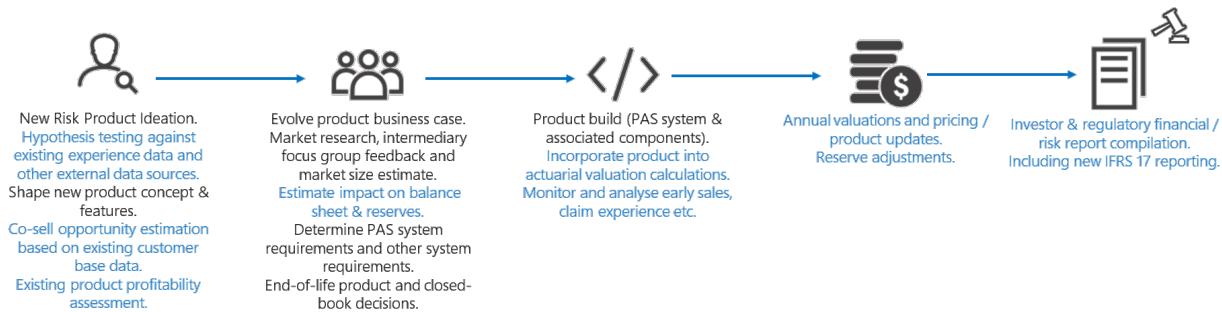
The following are some additional ideas that you can use as a starting point for your finance solution.

- [Auditing, risk, and compliance management](#)
- [Business Process Management](#)
- [HPC System and Big Compute Solutions](#)
- [HPC Risk Analysis Template](#)
- [Loan ChargeOff Prediction with Azure HDInsight Spark Clusters](#)
- [Loan ChargeOff Prediction with SQL Server](#)
- [Loan Credit Risk + Default Modeling](#)
- [Loan Credit Risk with SQL Server](#)
- [Unlock Legacy Data with Azure Stack](#)

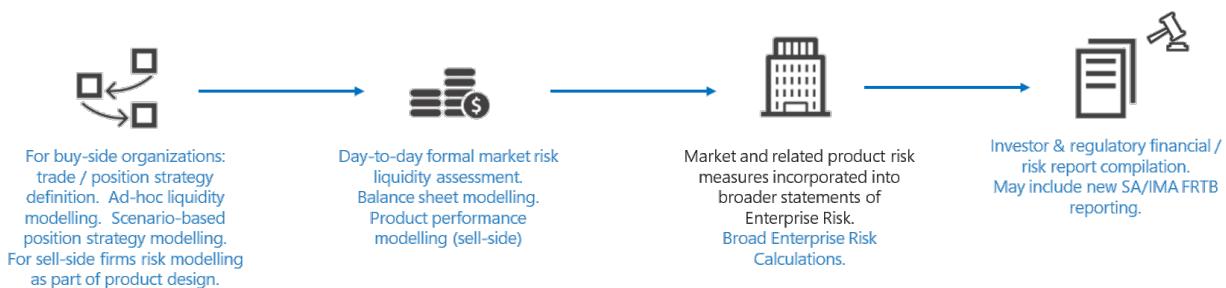
Enable the financial services risk lifecycle with Azure and R

3/10/2022 • 14 minutes to read • [Edit Online](#)

Risk calculations are pivotal at several stages in the lifecycle of key financial services operations. For example, a simplified form of the insurance product management lifecycle might look something like the diagram below. The risk calculation aspects are shown in blue text.



A scenario in a capital markets firm might look like this:



Through these processes, there are common needs around risk modeling including:

- The need for ad-hoc risk-related experimentation by risk analysts; actuaries in an insurance firm or quants in a capital markets firm. These analysts typically work with code and modeling tools popular in their domain: R and Python. Many university curriculums include training in R or Python in mathematical finance and in MBA courses. Both languages offer a wide range of open source libraries that support popular risk calculations.
- Along with appropriate tooling, analysts often require access to:
 - Accurate market pricing data.
 - Existing policy and claims data.
 - Existing market position data.
 - Other external data. Sources include structured data such as mortality tables and competitive pricing data. Less traditional sources such as weather, news and others may also be used.
 - Computational capacity to enable quick interactive data investigations.
- They may also make use of ad-hoc machine learning algorithms for pricing or determining market strategy.
- The need to visualize and present data for use in product planning, trading strategy, and similar discussions.

- The rapid execution of defined models, configured by the analysts for pricing, valuations, and market risk. The valuations use a combination of dedicated risk modeling, market risk tools, and custom code. The analysis is executed in a batch with varying nightly, weekly, monthly, quarterly, and annual calculations. This analysis generates spikes in workloads.
- The integration of data with other enterprise wide risk measures for consolidated risk reporting. In larger organizations, lower level risk estimates can be transferred to an enterprise risk modeling and reporting tool.
- Results must be reported in a defined format at the required interval to meet investor and regulatory requirements.

Microsoft supports the above concerns through a combination of Azure services and partner offerings in the [Azure Marketplace](#). In this article, we show practical examples of how to perform ad-hoc experimentation using R. We begin by explaining how to run the experiment on a single machine. Next, we show you how to run the same experiment on [Azure Batch](#), and we close by showing you how to take advantage of external services in our modeling. The options and considerations for the execution of defined models on Azure are described in these articles focused on [banking](#) and [insurance](#).

Analyst modeling in R

Let's start by looking at how R may be used by an analyst in a simplified, representative capital markets scenario. You can build this either by referencing an existing R library for the calculation or by writing code from scratch. In our example, we also must fetch external pricing data. To keep the example simple but illustrative, we calculate the potential future exposure (PFE) of an equity stock forward contract. This example avoids complex quantitative modeling techniques for instruments like complex derivatives and focuses on a single risk factor to concentrate on the risk life cycle. Our example lets you do the following actions:

- Select an instrument of interest.
- Source historic prices for the instrument.
- Model equity price by simple Monte Carlo (MC) calculation, which uses Geometric Brownian Motion (GBM):
 - Estimate expected return μ (mu) and volatility σ (theta).
 - Calibrate the model to historic data.
- Visualize the various paths to communicate the results.
- Plot $\max(0, \text{Stock Value})$ to demonstrate the meaning of PFE, the difference to Value at Risk (VaR).
 - To clarify: PFE = Share Price (T) -- Forward Contract Price K
- Take the 0.95 Quantile to get the PFE value at each time step / end of simulation period.

We'll calculate the potential future exposure for an equity forward based on Microsoft (MSFT) stock. As mentioned previously, to model the stock prices, historic prices for the MSFT stock are required so we can calibrate the model to historical data. There are many ways to acquire historical stock prices. In our example, we use a free version of a stock price service from an external service provider, [Quandl](#).

NOTE

The example uses the [WIKI Prices dataset](#) which can be used for learning concepts. For production usage of US based equities, Quandl recommends using the [End of Day US Stock Prices dataset](#).

To process the data and define the risk associated with the equity, we need to do the following things:

- Retrieve history data from the equity.
- Determine the expected return μ and volatility σ from the historic data.
- Model the underlying stock prices using some simulation.
- Run the model.
- Determine the exposure of the equity in the future.

We start by retrieving the stock from the Quandl service and plotting the closing price history over the last 180 days.

```
# Lubridate package must be installed
if (!require(lubridate)) install.packages('lubridate')
library(lubridate)

# Quandl package must be installed
if (!require(Quandl)) install.packages('Quandl')
library(Quandl)

# Get your API key from quandl.com
quandl_api = "enter your key here"

# Add the key to the Quandl keychain
Quandl.api_key(quandl_api)

quandl_get <-
  function(sym, start_date = "2018-01-01") {
    require(devtools)
    require(Quandl)
    # Retrieve the Open, High, Low, Close and Volume Column for a given Symbol
    # Column Indices can be deduced from this sample call
    # data <- Quandl(c("WIKI/MSFT"), rows = 1)

    tryCatch(Quandl(c(
      paste0("WIKI/", sym, ".8"),     # Column 8 : Open
      paste0("WIKI/", sym, ".9"),     # Column 9 : High
      paste0("WIKI/", sym, ".10"),    # Column 10: Low
      paste0("WIKI/", sym, ".11"),    # Column 11: Close
      paste0("WIKI/", sym, ".12")),   # Column 12: Volume
      start_date = start_date,
      type = "raw"
    ))
  }

# Define the Equity Forward
instrument.name <- "MSFT"
instrument.premium <- 100.1
instrument.pfeQuantile <- 0.95

# Get the stock price for the last 180 days
instrument.startDate <- today() - days(180)

# Get the quotes for an equity and transform them to a data frame
df_instrument.timeSeries <- quandl_get(instrument.name,start_date = instrument.startDate)

#Rename the columns
colnames(df_instrument.timeSeries) <- c()
colnames(df_instrument.timeSeries) <- c("Date","Open","High","Low","Close","Volume")

# Plot the closing price history to get a better feeling for the data
plot(df_instrument.timeSeries$Date, df_instrument.timeSeries$Close)
```

With the data in hand, we calibrate the GBM model.

```

# Code inspired by the book Computational Finance, An Introductory Course with R by
#     A. Arratia.

# Calculate the daily return in order to estimate sigma and mu in the Wiener Process
df_instrument.dailyReturns <- c(diff(log(df_instrument.timeSeries$Close)), NA)

# Estimate the mean of std deviation of the log returns to estimate the parameters of the Wiener Process

estimateGBM_Parameters <- function(logReturns, dt = 1/252) {

    # Volatility
    sigma_hat = sqrt(var(logReturns)) / sqrt(dt)

    # Drift
    mu_hat = mean(logReturns) / dt + sigma_hat**2 / 2.0

    # Return the parameters
    parameter.list <- list("mu" = mu_hat, "sigma" = sigma_hat)

    return(parameter.list)
}

# Calibrate the model to historic data
GBM_Parameters <- estimateGBM_Parameters(df_instrument.dailyReturns[1:length(df_instrument.dailyReturns) - 1])

```

Next, we model the underlying stock prices. We can either implement the discrete GBM process from scratch or utilize one of many R packages which provide this functionality. We use the R package [sde \(Simulation and Inference for Stochastic Differential Equations\)](#) which provides a method of solving this problem. The GBM method requires a set of parameters which are either calibrated to historic data or given as simulation parameters. We use the historic data, providing μ , σ and the stock prices at the beginning of the simulation (P_0).

```

if (!require(sde)) install.packages('sde')
library(sde)

sigma <- GBM_Parameters$sigma
mu <- GBM_Parameters$mu
P0 <- tail(df_instrument.timeSeries$Close, 1)

# Calculate the PFE looking one month into the future
T <- 1 / 12

# Consider nt MC paths
nt=50

# Divide the time interval T into n discrete time steps
n = 2 ^ 8

dt <- T / n
t <- seq(0,T,by=dt)

```

We're now ready to start a Monte Carlo simulation to model the potential exposure for some number of simulation paths. We'll limit the simulation to 50 Monte Carlo paths and 256 time steps. In preparation for scaling out the simulation and taking advantage of parallelization in R, the Monte Carlo simulation loop uses a `foreach` statement.

```

# Track the start time of the simulation
start_s <- Sys.time()

# Instead of a simple for loop to execute a simulation per MC path, call the
# simulation with the foreach package
# in order to demonstrate the similarity to the AzureBatch way to call the method.

library(foreach)
# Execute the MC simulation for the wiener process utilizing the GBM method from the sde package
exposure_mc <- foreach (i=1:nt, .combine = rbind ) %do% GBM(x = P0, r = mu, sigma = sigma, T = T, N = n)
rownames(exposure_mc) <- c()

# Track the end time of the simulation
end_s <- Sys.time()

# Duration of the simulation

difftime(end_s, start_s)

```

We've now simulated the price of the underlying MSFT stock. To calculate the exposure of the equity forward, we subtract the premium and limit the exposure to only positive values.

```

# Calculate the total Exposure as V_i(t) - K, put it to zero for negative exposures
pfe_mc <- pmax(exposure_mc - instrument.premium ,0)

ymax <- max(pfe_mc)
ymin <- min(pfe_mc)
plot(t, pfe_mc[1,], t = 'l', ylim = c(ymin, ymax), col = 1, ylab="Credit Exposure in USD", xlab="time t in
Years")
for (i in 2:nt) {
  lines(t, pfe_mc[i,], t = 'l', ylim = c(ymin, ymax), col = i)
}

```

The next two pictures show the result of the simulation. The first picture shows the Monte Carlo simulation of the underlying stock price for 50 paths. The second picture illustrates the underlying credit exposure for the equity forward after subtracting the premium of the equity forward and limiting the exposure to positive values.

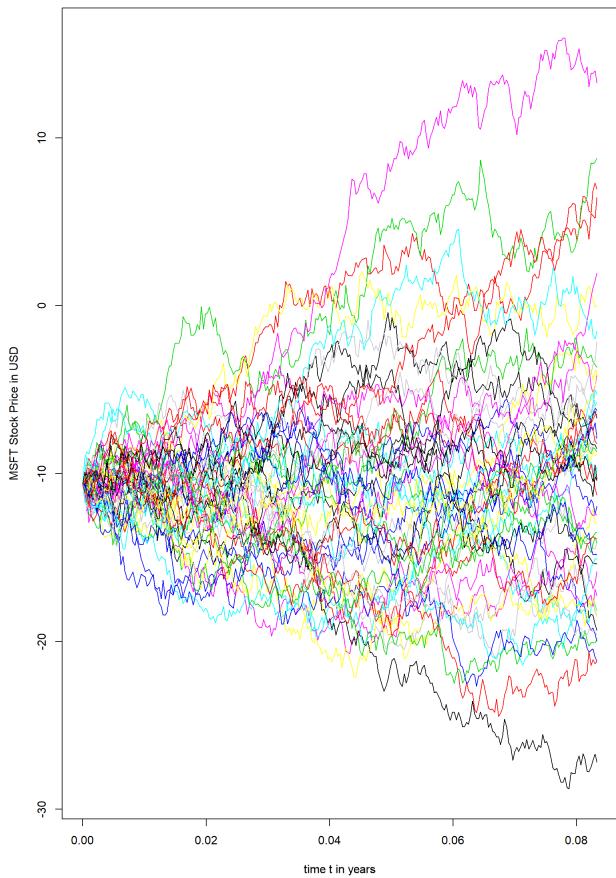


Figure 1 - 50 Monte Carlo paths

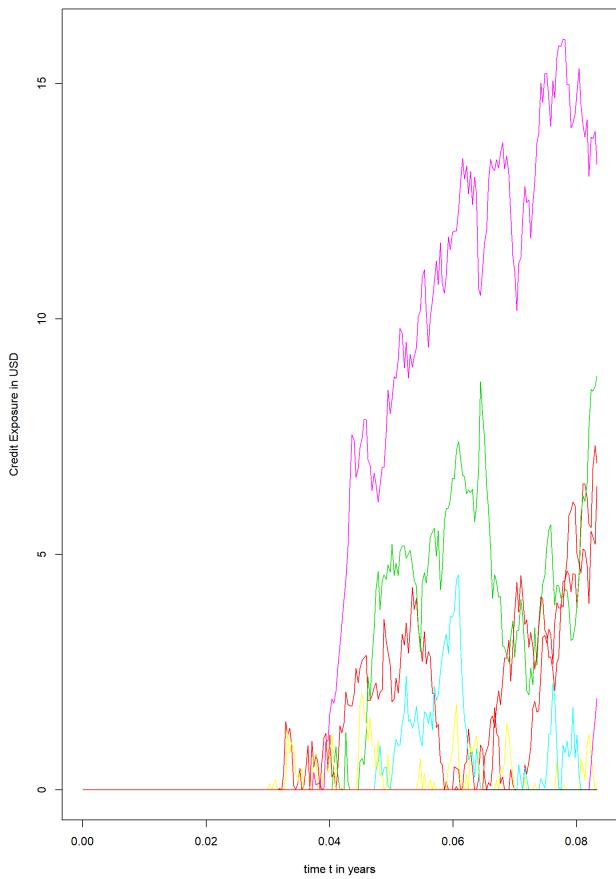


Figure 2 - Credit exposure for equity forward

In the last step, the 1-month 0.95 quantile PFE is calculated by the following code.

```

# Calculate the PFE at each time step
df_pfe <- cbind(t,apply(pfe_mc,2,quantile,probs = instrument.pfeQuantile ))

resulting in the final PFE plot
plot(df_pfe, t = 'l', ylab = "Potential Future Exposure in USD", xlab = "time t in Years")

```

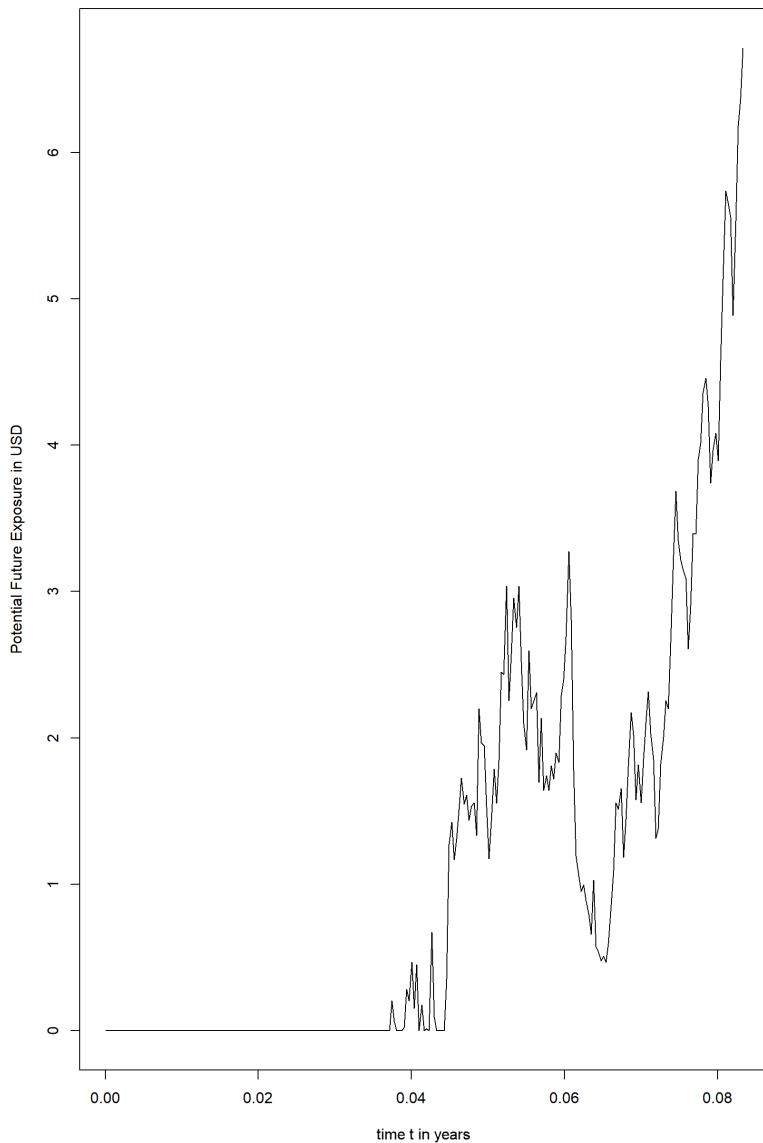


Figure 3 Potential future exposure for MSFT equity forward

Using Azure Batch with R

The R solution described above can be connected to Azure Batch and leverage the cloud for risk calculations. This takes little extra effort for a parallel calculation such as ours. The tutorial, [Run a parallel R simulation with Azure Batch](#), provides detailed information on connecting R to Azure Batch. Below we show the code and summary of the process to connect to Azure Batch and how to take advantage of the extension to the cloud in a simplified PFE calculation.

This example tackles the same model described earlier. As we've seen before, this calculation can run on our personal computer. Increases to the number of Monte Carlo paths or use of smaller time steps will result in much longer execution times. Almost all of the R code will remain unchanged. We'll highlight the differences in this section.

Each path of the Monte Carlo simulation runs in Azure. We can do this because each path is independent of the

others, giving us an "embarrassingly parallel" calculation.

To use Azure Batch, we define the underlying cluster and reference it in the code before the cluster can be used in the calculations. To run the calculations, we use the following cluster.json definition:

```
{  
  "name": "myMCPool",  
  "vmSize": "Standard_D2_v2",  
  "maxTasksPerNode": 4,  
  "poolSize": {  
    "dedicatedNodes": {  
      "min": 1,  
      "max": 1  
    },  
    "lowPriorityNodes": {  
      "min": 3,  
      "max": 3  
    },  
    "autoscaleFormula": "QUEUE"  
  },  
  "containerImage": "rocker/tidyverse:latest",  
  "rPackages": {  
    "cran": [],  
    "github": [],  
    "bioconductor": []  
  },  
  "commandLine": [],  
  "subnetId": ""  
}
```

With this cluster definition, the following R code makes use of the cluster:

```
# Define the cloud burst environment  
library(doAzureParallel)  
  
# set your credentials  
setCredentials("credentials.json")  
  
# Create your cluster if not exist  
cluster <- makeCluster("cluster.json")  
  
# register your parallel backend  
registerDoAzureParallel(cluster)  
  
# check that your workers are up  
getDoParWorkers()
```

Finally, we update the foreach statement from earlier to use the doAzureParallel package. It's a minor change, adding a reference to the sde package and changing the %do% to %dopar%:

```
# Execute the MC simulation for the wiener process utilizing the GBM method from the sde package and extend  
the computation to the cloud  
exposure_mc <- foreach(i = 1:nt, .combine = rbind, .packages = 'sde') %dopar% GBM(x = P0, r = mu, sigma =  
sigma, T = T, N = n)  
rownames(exposure_mc) <- c()
```

Each Monte Carlo simulation is submitted as a task to Azure Batch. The task executes in the cloud. Results are merged before being sent back to the analyst workbench. The heavy lifting and computations execute in the cloud to take full advantage of scaling and the underlying infrastructure required by the requested calculations.

After the calculations have finished, the additional resources can easily be shut-down by invoking the following

a single instruction:

```
# Stop the cloud cluster  
stopCluster(cluster)
```

Use a SaaS offering

The first two examples show how to utilize local and cloud infrastructure for developing an adequate valuation model. This paradigm has begun to shift. In the same way that on-premises infrastructure has transformed into cloud-based IaaS and PaaS services, the modeling of relevant risk figures is transforming into a service-oriented process. Today's analysts face two major challenges:

- The regulatory requirements use increasing compute capacity to add to modeling requirements. The regulators are asking for more frequent and up-to date risk figures.
- The existing risk infrastructure has grown organically with time and creates challenges when implementing new requirements and more advanced risk modeling in an agile manner.

Cloud-based services can deliver the required functionality and support risk analysis. This approach has some advantages:

- The most common risk calculations required by the regulator must be implemented by everyone under the regulation. By utilizing services from a specialized service provider, the analyst benefits from ready to use, regulator-compliant risk calculations. Such services may include market risk calculations, counterparty risk calculations, X-Value Adjustment (XVA), and even Fundamental Review of Trading Book (FRTB) calculations.
- These services expose their interfaces through web services. The existing risk infrastructure can be enhanced by these other services.

In our example, we want to invoke a cloud-based service for FRTB calculations. Several of these can be found on [AppSource](#). For this article we chose a trial option from [Vector Risk](#). We'll continue to modify our system. This time, we use a service to calculate the risk figure of interest. This process consists of the following steps:

1. Call the relevant risk service and with the right parameters.
2. Wait until the service finishes the calculation.
3. Retrieve and incorporate the results into the risk analysis.

Translated into R code, our R code can be enhanced by the definition of the required input values from a prepared input template.

```
Template <- readLines('RequiredInputData.json')  
data <- list()  
# drilldown setup  
timeSteps = seq(0, n, by = 1),  
paths = as.integer(seq(0, nt, length.out = min(nt, 100))),  
# calc setup  
calcDate = instrument.startDate,  
npaths = nt,  
price = P0,  
vol = sigma,  
drift = mu,  
premium = instrument.premium,  
maturityDate = today()  
)  
body <- whisker.render(template, data)
```

Next, we need to call the web service. In this case, we call the StartCreditExposure method to trigger the calculation. We store the endpoint for the API in a variable named *endpoint*.

```
# make the call
result <- POST( paste(endpoint, "StartCreditExposure", sep = ""),
                 authenticate(username, password, type = "basic"),
                 content_type("application/json"),
                 add_headers(`Ocp-Apim-Subscription-Key` = api_key),
                 body = body, encode = "raw"
               )

result <- content(result)
```

Once the calculations have finished, we retrieve the results.

```
# get back high level results
result <- POST( paste(endpoint, "GetCreditExposureResults", sep = ""),
                 authenticate(username, password, type = "basic"),
                 content_type("application/json"),
                 add_headers(`Ocp-Apim-Subscription-Key` = api_key),
                 body = sprintf('{"getCreditExposureResults":',
                               {"token":"DataSource=Production;Organisation=Microsoft", "ticket": "%s"}), ticket), encode = "raw")

result <- content(result)
```

This leaves the analyst to continue with the results received. The relevant risk figures of interest are extracted from the results and plotted.

```
if (!is.null(result$error)) {
  cat(result$error$message)
} else {
  # plot PFE
  result <- result$getCreditExposureResultsResponse$getCreditExposureResultsResult
  df <- do.call(rbind, result$exposures)
  df <- as.data.frame(df)
  df <- subset(df, term <= n)
}

plot(as.numeric(df$term[df$statistic == 'PFE']) / 365, df$result[df$statistic == 'PFE'], type = "p", xlab =
  ("time t in Years"), ylab = ("Potential Future Exposure in USD"), ylim = range(c(df$result[df$statistic ==
  'PFE'], df$result[df$statistic == 'PFE'])), col = "red")
```

The resulting plots look like this:

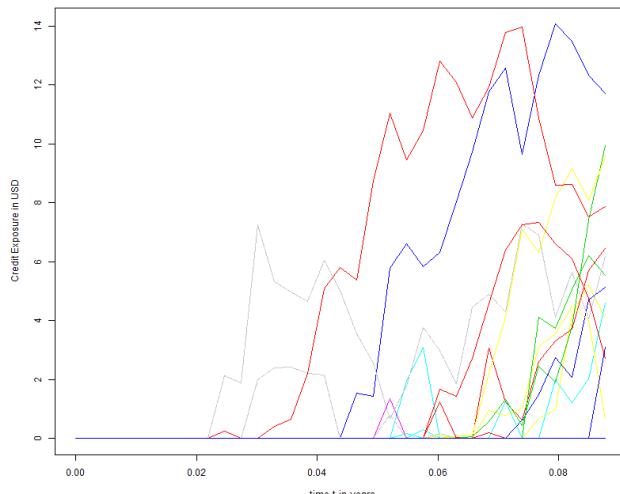


Figure 4 - Credit exposure for MSFT equity forward - Calculated with a cloud-based risk engine

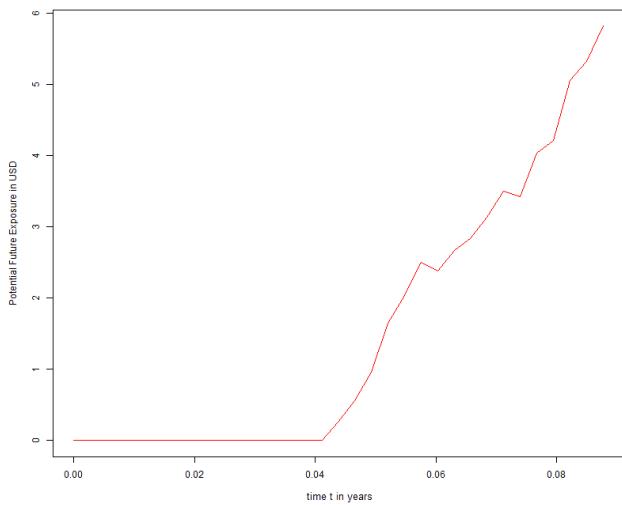


Figure 5 - Potential future exposure for MSFT equity forward - Calculated with a cloud-based risk engine

Contributors

This article is being updated and maintained by Microsoft. It was originally written by the following contributors:

- [Dr. Darko Mocelj](#) | HPC Global Blackbelt & AI Sr. Technology Specialist
- [Rupert Nicolay](#) | Financial Services Industry Solutions Lead

Next steps

Flexible access to the cloud through compute infrastructure and SaaS-based risk analysis services can deliver improvements in speed and agility for risk analysts working in capital markets and insurance. In this article we worked through an example which illustrates how to use Azure and other services using tools risk analysts know. Try taking advantage of Azure's capabilities as you create and enhance your risk models.

Tutorials

- R developers: [Run a parallel R simulation with Azure Batch](#)
- [Basic R commands and RevoScaleR functions: 25 common examples](#)
- [Visualize and analyze data using RevoScaleR](#)
- [Introduction to ML services and open-source R capabilities on HDInsight](#)

Solutions for the healthcare industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

The healthcare industry includes various systems that provide curative, preventative, rehabilitative, and palliative care to patients. Proper management of these systems enables healthcare providers and managers provide high-quality care and treatment for their patients. With Azure cloud and other Microsoft services, you can now create highly efficient and resilient healthcare systems that take care of not only the patient-provider interactions, but also provide clinical and data insights, leading to a more patient-centric strategy for the healthcare institute.

Modernization and digital transformation of healthcare facilities is all the more important during the current COVID-19 global pandemic.

Learn how you can use [Microsoft Azure](#) services to digitize, modernize, and enhance your healthcare solution at [Azure for healthcare](#). Microsoft also provides a comprehensive platform for the healthcare industry, [Microsoft Cloud for Healthcare](#), which includes components from [Dynamics 365](#) and [Microsoft 365](#), in addition to Azure.

Architectures for healthcare

The following articles provide detailed analysis of architectures developed and recommended for the healthcare industry.

Architecture	Summary	Technology Focus
Virtual health on Microsoft Cloud for Healthcare	Use Microsoft Cloud for Healthcare , a software package created for the healthcare industry, to build an architecture for scheduling and following up on virtual visits between patients, providers, and care managers.	Web
Clinical insights with Microsoft Cloud for Healthcare	Use Microsoft Cloud for Healthcare to collect, analyze, and visualize medical and health insights, that can be used to improve healthcare operations.	Web
Consumer health portal on Azure	Learn how to develop a consumer health portal using Azure services, to track statistics from wearables, engage with medical providers, and monitor health habits, built on a foundation of the Azure Well Architected Framework .	Web
Confidential computing for healthcare	Use Azure confidential computing to encrypt medical and patient data, for secure collaboration between hospitals and third-party diagnostic providers.	Security

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Health Data Consortium on Azure	Use the Azure Data Platform, and Azure Data Share to create an environment where healthcare organizations can appropriately, and securely share data with partner organizations to support activities like clinical trials and research.	Data
Precision Medicine Pipeline with Genomics	Use Microsoft Genomics and the Azure Data Platform to perform analysis and reporting for scenarios like precision medicine and genetic profiling.	Data/Analytics
Predict Hospital Readmissions with Machine Learning	Predict the readmissions of diabetic patients using Azure Data, AI, and Analytics tools through the different personas of Data Professionals throughout the process.	Data/AI
Build a telehealth system with Azure	Explore a customer's implementation of a telehealth system using Azure cloud services.	Containers

Solution ideas for healthcare

The following are some additional ideas that you can use as a starting point for your healthcare solution.

- [Population Health Management for Healthcare](#)
- [Medical Data Storage Solutions](#)
- [HIPAA and HITRUST compliant health data AI](#)
- [Remote Patient Monitoring Solutions](#)
- [Predict Length of Stay and Patient Flow](#)
- [Predict Length of Stay in Hospitals](#)
- [Contactless IoT interfaces with Azure intelligent edge](#)
- [COVID-19 Safe Solutions with IoT Edge](#)
- [IoT Connected Platform for COVID-19 detection and prevention](#)
- [UVEN smart and secure disinfection and lighting](#)

Solutions for the government industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

Microsoft Azure provides a mission-critical cloud platform, Azure Government, that delivers breakthrough innovation to US government customers and their partners. US federal, state, local, and tribal governments and their partners can have secure and dedicated access to this platform, with operations controlled by screened US citizens.

Using Azure Government, you can:

- test and deploy secure, highly-available, and performant mission-critical apps,
- create custom web experiences,
- gain insights from your data by using AI and analytics capabilities.

Azure Government offers a broad level of certifications to simplify critical government compliance requirements. To learn more about this government-focused cloud platform, visit [Azure Government](#).

Microsoft is committed to provide government agencies with innovative technology solutions across health and human services, critical infrastructure, public safety & justice, and tax, finance, and revenue. Learn more at [Cloud computing for government](#).

Architectures for government

The following articles provide detailed analysis of architectures developed and recommended for the government industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Azure Automation in a hybrid environment	Extend automated management and configuration from Azure to on-premises and other cloud providers.	Hybrid/Multicloud
Azure Automation Update Management	Design a hybrid update management solution to manage updates on both Microsoft Azure and on-premises Windows and Linux computers.	Hybrid/Multicloud
Computer forensics Chain of Custody in Azure	Ensure a valid Chain of Custody (CoC) in acquiring, storing, and accessing of digital evidence to support criminal investigations or civil proceedings.	Management/Governance
Hybrid Security Monitoring using Microsoft Defender for Cloud and Microsoft Sentinel	Monitor the security configuration and telemetry of on-premises and Azure operating system workloads.	Hybrid/Multicloud

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Vision classifier model with Azure Custom Vision Cognitive Service	Combine AI and Internet of Things (IoT) by using Azure Custom Vision to classify images taken by a simulated drone.	AI/ML
Web app private connectivity to Azure SQL database	Set up private connectivity from an Azure Web App to Azure Platform-as-a-Service (PaaS) services.	Security
Azure Virtual Desktop for the enterprise	Use Azure Virtual Desktop to build virtualized desktop infrastructure (VDI) solutions at enterprise scale, covering 1,000 virtual desktops and above.	Hybrid/Multicloud

Solution ideas for government

The following article provides an idea that you can use as a starting point for your government solution.

- [DevSecOps in GitHub](#)

Solutions for the manufacturing industry

3/10/2022 • 3 minutes to read • [Edit Online](#)

Manufacturing sector, a hallmark of the modern industrialized world, encompasses all steps from procuring raw materials to transforming into final product. Starting from household manufacturing in the pre-industrial era, this sector has evolved through stages such as mechanized assembly lines and automation, every new development adding to faster and more efficient manufacturing processes. Cloud computing can bring forth the next revolution for manufacturing companies by transforming their IT infrastructures and processes from error-prone on-premises to highly available, secure, and efficient cloud, as well as providing cutting edge Internet of Things (IoT), AI/ML, and analytics solutions.

Microsoft Azure holds the promise of the [fourth industrial revolution](#) by providing manufacturing solutions that can do the following:

- Help build more agile smart factories with industrial IoT.
- Create more resilient and profitable supply chains.
- Transform your work force productivity.
- Unlock innovation and new business models.
- Engage with customers in new ways.

To learn how you can modernize your manufacturing business using Azure, visit [Azure for manufacturing](#). For more resources, see [Microsoft Trusted Cloud for Manufacturing](#).

Architecture guides for manufacturing

The following articles provide architectural guidelines for Azure solutions in the manufacturing industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Azure industrial IoT analytics guidance	Build an architecture for an Industrial IoT (IIoT) analytics solution on Azure using PaaS (Platform as a service) components.	IoT
Upscale machine learning lifecycle with MLOps framework	Learn how a Fortune 500 food company improved its demand forecasting and optimized the product stocks in different stores across several regions in US with the help of customized machine learning models.	AI/ML
On-demand, scalable, high-power compute	In this article, we walk through some well-known areas in engineering and manufacturing that need large computing power and explore how the Microsoft Azure platform can help.	Compute

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Predictive maintenance in manufacturing	After introducing some background to predictive maintenance, we discuss how to implement the various pieces of a PdM solution using a combination of on-premises data, Azure machine learning, and usage of the machine learning models.	AI/ML
Predictive maintenance solution	This article presents options for building a predictive maintenance solution. It presents different perspectives and reference existing materials to get you started.	AI/ML
Extract actionable insights from IoT data	This guide provides a technical overview of the components needed to extract actionable insights from IoT data analytics.	IoT

Architectures for manufacturing

The following articles provide detailed analysis of architectures developed and recommended for the manufacturing industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Anomaly detector process	The Anomaly Detector API enables you to monitor and detect abnormalities in your time series data without having to know machine learning.	Analytics
Automated guided vehicles fleet control	This example architecture shows an end-to-end approach for an automotive original equipment manufacturer (OEM) and includes a reference architecture and several published supporting open-source libraries that can be reused.	IoT
Build a speech-to-text transcription pipeline with Azure Cognitive Services	Improve the efficiency of your customer care centers and transform your business by analyzing high volumes of recorded calls and building a speech-to-text transcription pipeline with Azure Cognitive Services.	AI/ML
Citizen AI with the Power Platform	The architecture extends on the Analytics end-to-end with Azure Synapse scenario. It allows for a custom ML model to be trained in Azure Machine Learning, and implemented with a custom application built using Microsoft Power Platform.	AI/ML

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Connected factory hierarchy service	A hierarchy service allows your business stakeholders to centrally define how production assets like machines are organized within factories, from both an operational and maintenance point of view.	IoT
End-to-end manufacturing using computer vision on the edge	This example architecture shows an end-to-end approach to computer vision from the edge to the cloud and back.	AI/ML
Optimized storage – time based - multi writes	This architecture uses multiple storage services to optimize storage performance and cost.	Databases
Predictive maintenance with the intelligent IoT Edge	The Internet-of-things (IoT) Edge brings data processing and storage close to the data source, enabling fast, consistent responses with reduced dependency on cloud connectivity and resources.	IoT
Quality assurance	This solution shows how to predict failures using the example of manufacturing pipelines (assembly lines).	AI/ML

Solution ideas for manufacturing

The following are other ideas that you can use as a starting point for your manufacturing solution.

- [Condition monitoring for industrial IoT](#)
- [COVID-19 safe environments with IoT Edge monitoring and alerting](#)
- [Create personalized marketing solutions in near real time](#)
- [Defect prevention with predictive maintenance](#)
- [Demand forecasting](#)
- [Demand forecasting for shipping and distribution](#)
- [Environment monitoring and supply chain optimization with IoT](#)
- [Facilities management powered by mixed reality and IoT](#)
- [Image classification with convolutional neural networks](#)
- [Knowledge mining for customer support and feedback analysis](#)
- [Low-latency network connections for industry](#)
- [Predictive aircraft engine monitoring](#)
- [Predictive insights with vehicle telematics](#)
- [Predictive maintenance](#)
- [Predictive marketing with machine learning](#)
- [Supply chain track and trace](#)

Solutions for the media and entertainment industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

The media and entertainment industry captures one of the largest market shares. It is comprised of businesses that produce and distribute content, such as motion pictures, television programs and commercials, streaming content, music and audio recordings, radio, book publishing, video games, and so on. With the COVID-19 pandemic greatly impacting and accelerating shifts in consumer behaviors, this industry is seeing trends such as creating more virtual, streamed, and personal content. It is all the more important for media businesses to harness the power of cloud computing and reach their customers in more personalized and innovative ways.

Microsoft's Azure and other offerings are committed to empower media and entertainment businesses to achieve more:

- accelerate content creation,
- provide cost-effective content management platforms,
- optimize and personalize content delivery,
- modernize collaboration.

To learn how Azure can provide an intelligent cloud backbone to content owners and creators, visit [Azure for media and entertainment](#). Microsoft offerings are transforming and empowering media businesses; see some case studies at [Intelligent Media and Entertainment](#).

Architectures for media and entertainment

The following articles provide detailed analysis of architectures developed and recommended for the media and entertainment industry.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
3D video rendering with Azure Batch	Use Azure Batch as a powerful yet cost-effective alternative to expensive high end computing resources, for 3D video rendering.	Compute
Digital image-based modeling on Azure	Build a high-performance and scalable image-based modeling architecture using Azure infrastructure-as-a-service (IaaS).	Compute
Image classification on Azure	Quickly build an architecture to process, classify, and analyze images, using Azure Computer Vision API.	AI/ML
Movie recommendations on Azure	Automate movie and product recommendations by using an Azure Data Science Virtual Machine to train an Azure Machine Learning model.	AI/ML

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Scalable personalization on Azure	Build a <i>content-based</i> personalized recommendation system for customers by learning their preferences from past purchases and interactions.	AI/ML
Analyze news feeds with near real-time analytics	Build a pipeline for mass ingestion and near real-time analysis of documents coming from public RSS news feeds using Azure services.	Analytics

Solution ideas for media and entertainment

The following are other ideas that you can use as a starting point for your media and entertainment solution.

- [Knowledge mining in digital asset management](#)
- [HPC Media Rendering](#)
- [Suggest content tags with deep learning and NLP](#)
- [Keyword search/speech-to-text/OCR digital media](#)

Solutions for the energy and environment industries

3/10/2022 • 2 minutes to read • [Edit Online](#)

Rising energy needs and sustainability targets are pushing companies to explore innovative solutions and architectures. Innovations like IoT, AI, and machine learning can help address these critical needs.

Power and utilities solutions can help industries build a future of optimized energy management and sustainability and respond to climate change.

Learn how you can join the global community committed to reducing energy use, transitioning to a more carbon-neutral grid, and promoting a greener future with sustainable and innovative technologies from Azure.

Architectures for energy and the environment

The following articles provide detailed analysis of architectures developed and recommended for the energy and environment industries.

Architecture	Summary	Technology Focus
Cost savings through HTAP with Azure SQL	Use a scalable hybrid transaction/analytical processing (HTAP) architecture with Azure SQL Database to combine transaction processing with analytics. For example, energy providers can use this architecture to manage smart power grids.	Data/Analytics
Create smart places by using Azure Digital Twins	Use Azure Digital Twins to create models of smart places from IoT device data. View and control products, systems, environments, and experiences to optimize energy.	IoT
Efficient Docker image deployment for intermittent low-bandwidth connectivity	Learn about a reliable and resilient deployment architecture for situations when you have limited, intermittent, or low bandwidth. This architecture is applicable in fields like the oil, gas, and mining industries.	IoT
Environment monitoring and supply chain optimization with IoT	Learn how to use Azure IoT for environment monitoring and supply chain optimization. Use cases include fire prediction and farming.	IoT

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Geospatial data processing and analytics	Collect, process, and store geospatial data by using managed Azure services. Make the data available through web apps. Visualize, explore, and analyze the data. You can use this architecture for applications like processing and storing climate data.	Data/Analytics
Mining equipment monitoring	Send equipment data to the Azure cloud to monitor its condition and performance. Process the data by using Azure Databricks, and alert operators and administrators as needed.	Data/Analytics
Project 15 Open Platform IoT sustainability	Use IoT technologies and the Project 15 Open Platform to accelerate innovation in species tracking, ecosystem monitoring, and other areas.	IoT
Run CFD simulations	Learn about running computational fluid dynamics (CFD) simulations for applications in the oil and gas industries, for example. Create, manage, and optimize clusters by using Azure CycleCloud.	Compute
Run reservoir simulation software on Azure	Run OPM Flow reservoir simulation software and OPM ResInsight visualization software on an Azure HPC compute cluster and visualization VM. You can use this architecture for applications like 3D reservoir modeling and visualization of seismic data.	Compute
Scale AI and machine learning initiatives in regulated industries	Learn about scaling Azure AI and machine learning environments that need to comply with extensive security policies, like those used in the oil and gas industries.	AI / Machine learning

Solution ideas for energy and the environment

The following are other ideas that you can use as a starting point for your energy or environment solution.

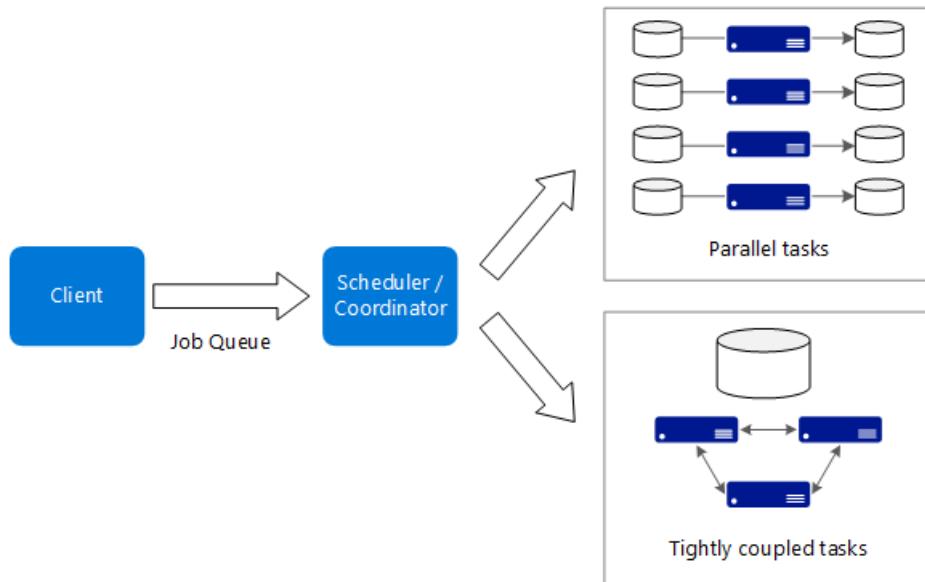
- [Azure IoT Edge data storage and processing](#)
- [Data science and machine learning with Azure Databricks](#)
- [Demand forecasting](#)
- [Energy supply optimization](#)
- [Forecast energy and power demand](#)
- [IoT analyze and optimize loops](#)
- [IoT-connected light, power, and internet for emerging markets](#)
- [IoT device connectivity for healthcare facilities](#)
- [IoT monitor and manage loops](#)

- Low-latency network connections for industry
- Modern analytics architecture with Azure Databricks
- Oil and gas tank-level forecasting
- Stromasys Charon-SSP Solaris emulator on Azure VMs

Big compute architecture style

3/10/2022 • 3 minutes to read • [Edit Online](#)

The term *big compute* describes large-scale workloads that require a large number of cores, often numbering in the hundreds or thousands. Scenarios include image rendering, fluid dynamics, financial risk modeling, oil exploration, drug design, and engineering stress analysis, among others.



Here are some typical characteristics of big compute applications:

- The work can be split into discrete tasks, which can be run across many cores simultaneously.
- Each task is finite. It takes some input, does some processing, and produces output. The entire application runs for a finite amount of time (minutes to days). A common pattern is to provision a large number of cores in a burst, and then spin down to zero once the application completes.
- The application does not need to stay up 24/7. However, the system must handle node failures or application crashes.
- For some applications, tasks are independent and can run in parallel. In other cases, tasks are tightly coupled, meaning they must interact or exchange intermediate results. In that case, consider using high-speed networking technologies such as InfiniBand and remote direct memory access (RDMA).
- Depending on your workload, you might use compute-intensive VM sizes (H16r, H16mr, and A9).

When to use this architecture

- Computationally intensive operations such as simulation and number crunching.
- Simulations that are computationally intensive and must be split across CPUs in multiple computers (10-1000s).
- Simulations that require too much memory for one computer, and must be split across multiple computers.
- Long-running computations that would take too long to complete on a single computer.
- Smaller computations that must be run 100s or 1000s of times, such as Monte Carlo simulations.

Benefits

- High performance with "[embarrassingly parallel](#)" processing.
- Can harness hundreds or thousands of computer cores to solve large problems faster.

- Access to specialized high-performance hardware, with dedicated high-speed InfiniBand networks.
- You can provision VMs as needed to do work, and then tear them down.

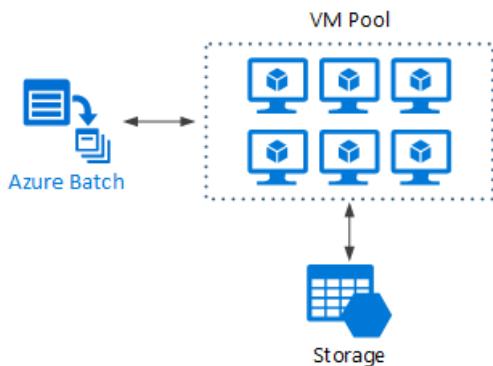
Challenges

- Managing the VM infrastructure.
- Managing the volume of number crunching
- Provisioning thousands of cores in a timely manner.
- For tightly coupled tasks, adding more cores can have diminishing returns. You may need to experiment to find the optimum number of cores.

Big compute using Azure Batch

[Azure Batch](#) is a managed service for running large-scale high-performance computing (HPC) applications.

Using Azure Batch, you configure a VM pool, and upload the applications and data files. Then the Batch service provisions the VMs, assign tasks to the VMs, runs the tasks, and monitors the progress. Batch can automatically scale out the VMs in response to the workload. Batch also provides job scheduling.

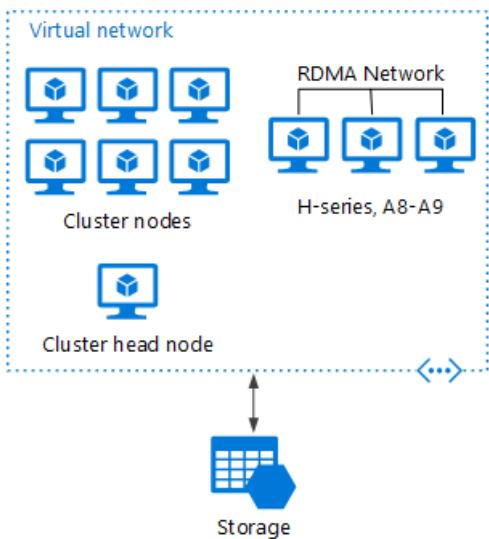


Big compute running on Virtual Machines

You can use [Microsoft HPC Pack](#) to administer a cluster of VMs, and schedule and monitor HPC jobs. With this approach, you must provision and manage the VMs and network infrastructure. Consider this approach if you have existing HPC workloads and want to move some or all it to Azure. You can move the entire HPC cluster to Azure, or you can keep your HPC cluster on-premises but use Azure for burst capacity. For more information, see [Batch and HPC solutions for large-scale computing workloads](#).

HPC Pack deployed to Azure

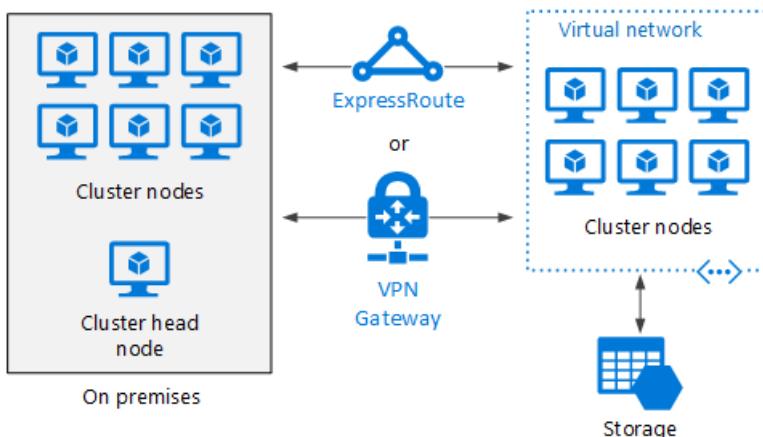
In this scenario, the HPC cluster is created entirely within Azure.



The head node provides management and job scheduling services to the cluster. For tightly coupled tasks, use an RDMA network that provides very high bandwidth, low latency communication between VMs. For more information, see [Deploy an HPC Pack 2016 cluster in Azure](#).

Burst an HPC cluster to Azure

In this scenario, an organization is running HPC Pack on-premises, and uses Azure VMs for burst capacity. The cluster head node is on-premises. ExpressRoute or VPN Gateway connects the on-premises network to the Azure VNet.



Next steps

- [Choose an Azure compute service for your application](#)
- [High Performance Computing \(HPC\) on Azure](#)
- [HPC cluster deployed in the cloud](#)

Solutions for the game development industry

3/10/2022 • 3 minutes to read • [Edit Online](#)

There are 2 billion gamers in the world today. They play a broad range of games, on a broad range of devices. Game creators strive to continuously engage players, spark their imaginations, and inspire them. Microsoft tools and services can help you achieve these goals.

Build, scale, and operate your game on the global, reliable Azure cloud, and incorporate features like multiplayer, leaderboards, translation, and bots. The following video shows how Azure can help bring multiplayer matchmaking into your game.

Architectures for game development

The following articles provide detailed analysis of architectures created and recommended for the game development industry.

AI in games

ARCHITECTURE	SUMMARY
Content moderation	Learn how to moderate content to maintain a civil, welcoming, and pleasurable experience among players.
Customer service bot for gaming	Create a conversational assistant that's tailored to your game and that understands natural language.
Image classification	Use Azure services like the Computer Vision API and Azure Functions to process images. For example, you could classify telemetry data from game screenshots.
Speech to text for gaming	Help bring everyone into the conversation by using the speech to text cognitive service provided by Azure.
Text to speech for gaming	Help bring everyone into the conversation by converting text messages to audio by using text to speech.
Text translation for gaming	Accommodate players in various languages by providing both the original message and a translation.

Analytics in games

ARCHITECTURE	SUMMARY
In-editor debugging telemetry	Gather data from gameplay sessions and display it directly within the game engine.
Non-real time analytics dashboard	Create a game analytics pipeline to use when you track data that doesn't require real-time analysis.

Databases for gaming

ARCHITECTURE	SUMMARY
Gaming using Azure MySQL	Elastically scale your Azure Database for MySQL database to accommodate unpredictable bursts of traffic and deliver low-latency multiplayer experiences on a global scale.
Gaming using Azure Cosmos DB	Elastically scale your Azure Cosmos DB database to accommodate unpredictable bursts of traffic and deliver low-latency multiplayer experiences on a global scale.

Game streaming

ARCHITECTURE	SUMMARY
Unreal Pixel Streaming	Deploy Unreal Engine's Pixel Streaming technology on Azure. You can use this Epic Games technology to stream remotely deployed interactive 3D applications through a browser.
Deploy Unreal Pixel Streaming	Deploy the Unreal Pixel Streaming package on an Azure GPU virtual machine or on multiple virtual machines.
Unreal Pixel Streaming at scale	Deploy Unreal Engine's Pixel Streaming technology at scale on Azure.

Leaderboards

ARCHITECTURE	SUMMARY
Leaderboard basics	Implement a leaderboard that suits your game design.
Non-relational leaderboard	Implement a gaming leaderboard that uses Azure Cache for Redis together with another database to improve data throughput and reduce database load.
Relational leaderboard	Enable a leaderboard in your large-scale game by using a relational database.

Matchmaking

ARCHITECTURE	SUMMARY
Multiplayer matchmaker	Build a multiplayer matchmaker by using serverless Azure functions.
Serverless matchmaker	Build a serverless multiplayer matchmaker that uses Azure Traffic Manager, Azure Functions, and Azure Event Hubs.

Rendering

ARCHITECTURE	SUMMARY
3D video rendering	Use Azure Batch to run large-scale 3D video rendering jobs.
Digital image-based modeling	Perform image-based modeling for your game's visual effects.

Scalable gaming servers

ARCHITECTURE	SUMMARY
Asynchronous multiplayer	Build an asynchronous multiplayer by saving game state to a persistent database.
Custom game server scaling	Containerize your game server with Docker and build a reliable, automated deployment process for servers by using Azure Resource Manager templates, Azure Functions, and DevOps practices.
Multiplayer backend reference architectures	Learn about a variety of multiplayer backend use cases and implementations that can help you create a cloud solution that works for your game.
Multiplayer hosting with Azure Batch	Build a scalable game server that's hosted on Azure Batch.
Multiplayer hosting with Service Fabric	Build a scalable game server that's hosted on Azure Service Fabric.
Multiplayer with Azure Container Instances	Learn about a multiplayer solution that automatically scales on demand and is billed per seconds of usage.
Multiplayer with Azure Kubernetes Service	Manage containerized, dedicated game servers by using the Kubernetes orchestrator on Azure.
Serverless asynchronous multiplayer	Build a serverless asynchronous multiplayer game on Azure.

Server hosting

ARCHITECTURE	SUMMARY
Basic game server hosting	Set up a basic Azure back end that hosts a game server on either Windows or Linux.
LAMP architectures for gaming	Learn how to effectively and efficiently deploy an existing LAMP architecture on Azure.

Related resources

- [Browse all our game development architectures](#)

Solutions for the travel and hospitality industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

Travel and hospitality are trillion-dollar industries in the United States alone. Travel and hospitality companies are investing in newer products and technologies, looking for that competitive edge. Cloud technology offers innovative, affordable, and versatile solutions. With integrated solutions, services, and templates, Azure gives developers and IT professionals the ability to create web solutions for the Internet of Things (IoT) and mobile devices.

Azure has the flexibility and scalability to handle the challenges that are faced in the travel and hospitality industries. Azure offers the following capabilities:

- Create hybrid environments by integrating your existing IT environment with the cloud, through secure private connections, hybrid databases, and storage solutions. Keep your assets where you want them.
- Increase or lower storage volume based on seasonal fluctuations and high-demand events. Scale to the demands of your travelers.
- Keep data secure and safe through encrypted solutions, firewalls, backup recovery, and redundancy.
- Build, deploy, and manage apps. Customize your cloud software to meet your needs.
- Improve customer service and uncover new business opportunities by using built-in support for analyzing data and key insights.

The following video shows how Azure AI can optimize your guest experience.

Architectures for travel and hospitality

The following articles provide detailed analysis of architectures created and recommended for the travel and hospitality industry.

Architecture	Summary	Technology Focus
Build a chatbot for hotel booking	Integrate a conversational chatbot into your applications. Chatbots can supplement your customer service agents, by responding to customer requests.	AI
Build a delta lake to support ad hoc queries in online leisure and travel booking	Make raw data easily accessible. This data is important to technical and customer support teams, data engineers, and legal teams.	Databases
Commerce chatbot for customer service	Create a conversational commerce chatbot for banking, travel, and entertainment.	AI
Custom business processes for airlines	Monitor customer flight data, when a traveler selects or is assigned a flight.	Integration

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Data science and machine learning with Azure Databricks	Improve efficiency, enhance customer experiences, and predict changes in your business with data science and machine learning.	AI
Migrate a web app using Azure API Management	Migrate from your legacy web applications by using Azure API Management.	Web
Predictive aircraft engine monitoring	Combine real-time aircraft data with analytics to create a solution for predictive aircraft engine monitoring and health.	Analytics

Solution ideas for travel and hospitality

The following are other ideas that you can use as a starting point for your travel and hospitality solution.

- [Build web and mobile applications](#)
- [Cognizant Safe Buildings with IoT and Azure](#)
- [Data science and machine learning with Azure Databricks](#)
- [Facilities management powered by mixed reality and IoT](#)
- [Predictive maintenance](#)

Data science using Spark on Azure HDInsight

3/10/2022 • 8 minutes to read • [Edit Online](#)

This suite of topics shows how to use HDInsight Spark to complete common data science tasks such as data ingestion, feature engineering, modeling, and model evaluation. The data used is a sample of the 2013 NYC taxi trip and fare dataset. The models built include logistic and linear regression, random forests, and gradient boosted trees. The topics also show how to store these models in Azure blob storage (WASB) and how to score and evaluate their predictive performance. More advanced topics cover how models can be trained using cross-validation and hyper-parameter sweeping. This overview topic also references the topics that describe how to set up the Spark cluster that you need to complete the steps in the walkthroughs provided.

Spark and MLlib

[Spark](#) is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for the iterative algorithms used in machine learning and graph computations. [MLlib](#) is Spark's scalable machine learning library that brings the algorithmic modeling capabilities to this distributed environment.

HDInsight Spark

[HDInsight Spark](#) is the Azure hosted offering of open-source Spark. It also includes support for [Jupyter PySpark notebooks](#) on the Spark cluster that can run Spark SQL interactive queries for transforming, filtering, and visualizing data stored in Azure Blobs (WASB). PySpark is the Python API for Spark. The code snippets that provide the solutions and show the relevant plots to visualize the data here run in Jupyter notebooks installed on the Spark clusters. The modeling steps in these topics contain code that shows how to train, evaluate, save, and consume each type of model.

Setup: Spark clusters and Jupyter notebooks

Setup steps and code are provided in this walkthrough for using an HDInsight Spark 1.6. But Jupyter notebooks are provided for both HDInsight Spark 1.6 and Spark 2.0 clusters. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked notebooks is generic and should work on any Spark cluster. If you are not using HDInsight Spark, the cluster setup and management steps may be slightly different from what is shown here. For convenience, here are the links to the Jupyter notebooks for Spark 1.6 (to be run in the pySpark kernel of the Jupyter Notebook server) and Spark 2.0 (to be run in the pySpark3 kernel of the Jupyter Notebook server):

Spark 1.6 notebooks

These notebooks are to be run in the pySpark kernel of Jupyter notebook server.

- [pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#): Provides information on how to perform data exploration, modeling, and scoring with several different algorithms.
- [pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): Includes topics in notebook #1, and model development using hyperparameter tuning and cross-validation.
- [pySpark-machine-learning-data-science-spark-model-consumption.ipynb](#): Shows how to operationalize a saved model using Python on HDInsight clusters.

Spark 2.0 notebooks

These notebooks are to be run in the pySpark3 kernel of Jupyter notebook server.

- [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): This file provides information on how to perform data exploration, modeling, and scoring in Spark 2.0 clusters using the NYC Taxi trip and fare data-set described [here](#). This notebook may be a good starting point for quickly exploring the code we have provided for Spark 2.0. For a more detailed notebook analyzes the NYC Taxi data, see the next notebook in this list. See the notes following this list that compares these notebooks.
- [Spark2.0-pySpark3_NYC_Taxi_Tip_Regression.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the NYC Taxi trip and fare data-set described [here](#).
- [Spark2.0-pySpark3_Airline_Departure_Delay_Classification.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the well-known Airline On-time departure dataset from 2011 and 2012. We integrated the airline dataset with the airport weather data (for example, windspeed, temperature, altitude etc.) prior to modeling, so these weather features can be included in the model.

NOTE

The airline dataset was added to the Spark 2.0 notebooks to better illustrate the use of classification algorithms. See the following links for information about airline on-time departure dataset and weather dataset:

- Airline on-time departure data: <https://www.transtats.bts.gov/ONTIME/>
- Airport weather data: <https://www.ncdc.noaa.gov/>

NOTE

The Spark 2.0 notebooks on the NYC taxi and airline flight delay data-sets can take 10 mins or more to run (depending on the size of your HDI cluster). The first notebook in the above list shows many aspects of the data exploration, visualization and ML model training in a notebook that takes less time to run with down-sampled NYC data set, in which the taxi and fare files have been pre-joined: [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#). This notebook takes a much shorter time to finish (2-3 mins) and may be a good starting point for quickly exploring the code we have provided for Spark 2.0.

For guidance on the operationalization of a Spark 2.0 model and model consumption for scoring, see the [Spark 1.6 document on consumption](#) for an example outlining the steps required. To use this example on Spark 2.0, replace the Python code file with [this file](#).

Prerequisites

The following procedures are related to Spark 1.6. For the Spark 2.0 version, use the notebooks described and linked to previously.

1. You must have an Azure subscription. If you do not already have one, see [Get Azure free trial](#).
 2. You need a Spark 1.6 cluster to complete this walkthrough. To create one, see the instructions provided in [Get started: create Apache Spark on Azure HDInsight](#). The cluster type and version is specified from the [Select Cluster Type](#) menu.

New HDInsight Cluster

*** Cluster Name**
Enter new cluster name .azurehdinsight.net

*** Subscription**
Azure-Irregulars_563702 >

Select Cluster Type **Spark (Preview)** (Configure required settings)

*** Credentials** (Configure required settings)

*** Data Source** (Configure required settings)

*** Node Pricing Tiers** (Configure required settings)

Cluster Type configuration

Learn about HDInsight and cluster versions. [Learn more](#)

Cluster Type Spark (Preview)

Operating System Linux

Spark 1.5.2 (HDI 3.3)

Spark 1.6.0 (HDI 3.4)

Cluster Tier (more info)	Price
STANDARD <ul style="list-style-type: none"> [Icon] Administration Manage, monitor, connect [Icon] Scalability On-demand node scaling [Icon] 99.9% Uptime SLA [Icon] Automatic patching 	+ 0.00 USD/CORE/HOUR
PREMIUM (PREVIEW) ★ <ul style="list-style-type: none"> [Icon] Administration Manage, monitor, connect [Icon] Scalability On-demand node scaling [Icon] 99.9% Uptime SLA [Icon] Automatic patching [Icon] Microsoft R Server for HDInsight 	+ 0.02 USD/CORE/HOUR

NOTE

For a topic that shows how to use Scala rather than Python to complete tasks for an end-to-end data science process, see the [Data Science using Scala with Spark on Azure](#).

WARNING

Billing for HDInsight clusters is prorated per minute, whether you use them or not. Be sure to delete your cluster after you finish using it. See [how to delete an HDInsight cluster](#).

The NYC 2013 Taxi data

The NYC Taxi Trip data is about 20 GB of compressed comma-separated values (CSV) files (~48 GB uncompressed), comprising more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and dropoff location and time, anonymized hack (driver's) license number and medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The 'trip_data' CSV files contain trip details, such as number of passengers, pick up and dropoff points, trip duration, and trip length. Here are a few sample records:

medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longit
89D227B655E5C82AEFC13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01 15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B678ED28BEA73D8,9FD8F69F0B040DBB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.750666
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0B040DBB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1.10,-74.004707,40.737777,-74.009834,40.726002
DFD2202EE08FTAB8DC9A57B02AC81FE2,51EE87E3205C985EF8431D0850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,,78,-73.974602,49.759945,-73.984734,40.759388
DFD2202EE08FTAB8DC9A57B02AC81FE2,51EE87E3205C985EF8431D0850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07 23:34:08,1,566,2,10,-73.97625,40.748528,-74.002586,40.747846

2. The 'trip_fare' CSV files contain details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

medallion, hack_license, vendor_id, pickup_datetime, payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, total_amount

89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01

OBP7C05FBAA12B906F0C77BFD20B5A73D0_0EB0E5C0E2004BDBEF4C5165F0BA1DE472_GMT_2012-01-20

00:18:35,CSH,6,0.5,0.5,0,0,7

0007C9F5BA12D886CB807BD28D8A/308,31,D81,031,0804BDBB5349140E5DA1BE4/2,CHI,2015-01-03
18:49:41,CSH,5.5,1,0.5,0,0,7

DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07
23:54:15 CSH 5 0 5 0 5 0 6

DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07

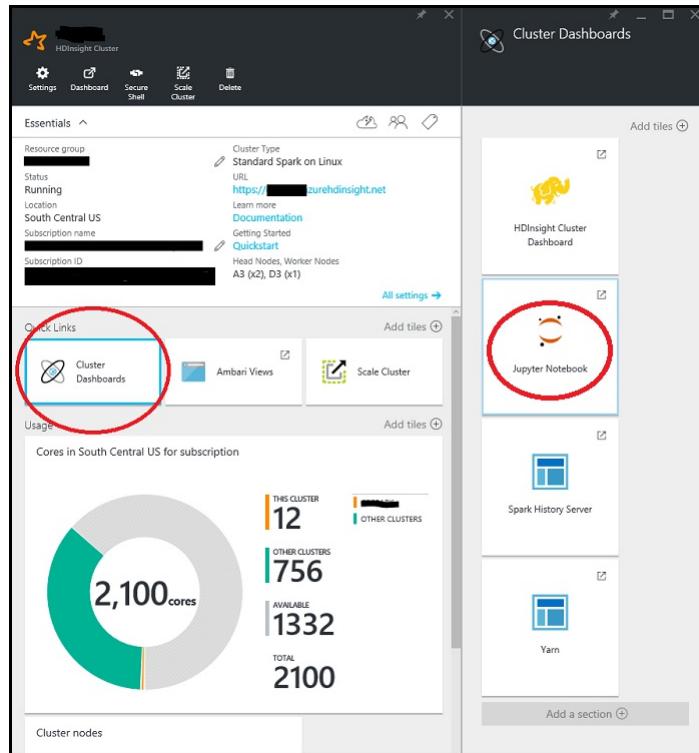
We have taken a 0.1% sample of these files and joined the trip_data and trip_fare CVS files into a single dataset to use as the input dataset for this walkthrough. The unique key to join trip_data and trip_fare is composed of the following fields: `id`, `start_time`, `end_time`, `start_station_id`, `end_station_id`, `duration`. Each record of the dataset contains the following:

attributes representing a NYC Taxi trip:

FIELD	BRIEF DESCRIPTION
medallion	Anonymized taxi medallion (unique taxi id)
hack_license	Anonymized Hackney Carriage License number
vendor_id	Taxi vendor id
rate_code	NYC taxi rate of fare
store_and_fwd_flag	Store and forward flag
pickup_datetime	Pick up date & time
dropoff_datetime	Dropoff date & time
pickup_hour	Pick up hour
pickup_week	Pick up week of the year
weekday	Weekday (range 1-7)
passenger_count	Number of passengers in a taxi trip
trip_time_in_secs	Trip time in seconds
trip_distance	Trip distance traveled in miles
pickup_longitude	Pick up longitude
pickup_latitude	Pick up latitude
dropoff_longitude	Dropoff longitude
dropoff_latitude	Dropoff latitude
direct_distance	Direct distance between pickup and dropoff locations
payment_type	Payment type (cash, credit-card etc.)
fare_amount	Fare amount in
surcharge	Surcharge
mta_tax	MTA Metro Transportation tax
tip_amount	Tip amount
tolls_amount	Tolls amount
total_amount	Total amount
tipped	Tipped (0/1 for no or yes)
tip_class	Tip class (0: \$0, 1: \$0-5, 2: \$6-10, 3: \$11-20, 4: > \$20)

Execute code from a Jupyter notebook on the Spark cluster

You can launch the Jupyter Notebook from the Azure portal. Find your Spark cluster on your dashboard and click it to enter management page for your cluster. To open the notebook associated with the Spark cluster, click **Cluster Dashboards -> Jupyter Notebook**.



You can also browse to <https://CLUSTERNAME.azurehdinsight.net/jupyter> to access the Jupyter Notebooks. Replace the CLUSTERNAME part of this URL with the name of your own cluster. You need the password for your admin account to access the notebooks.

Select PySpark to see a directory that contains a few examples of pre-packaged notebooks that use the PySpark API. The notebooks that contain the code samples for this suite of Spark topic are available at [GitHub](#)

You can upload the notebooks directly from [GitHub](#) to the Jupyter notebook server on your Spark cluster. On the home page of your Jupyter, click the **Upload** button on the right part of the screen. It opens a file explorer. Here you can paste the GitHub (raw content) URL of the Notebook and click Open.

You see the file name on your Jupyter file list with an **Upload** button again. Click this **Upload** button. Now you have imported the notebook. Repeat these steps to upload the other notebooks from this walkthrough.

TIP

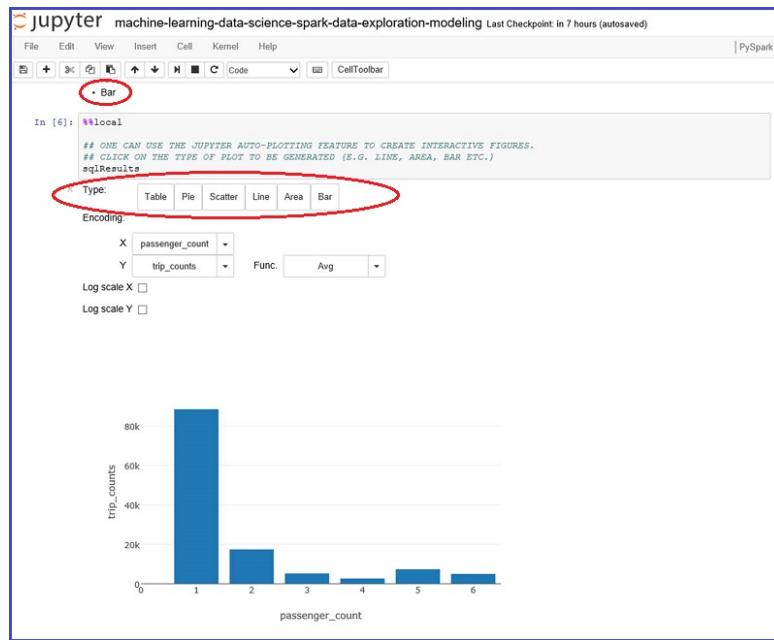
You can right-click the links on your browser and select **Copy Link** to get the GitHub raw content URL. You can paste this URL into the Jupyter Upload file explorer dialog box.

Now you can:

- See the code by clicking the notebook.
- Execute each cell by pressing **SHIFT-ENTER**.
- Run the entire notebook by clicking on **Cell -> Run**.
- Use the automatic visualization of queries.

TIP

The PySpark kernel automatically visualizes the output of SQL (HiveQL) queries. You are given the option to select among several different types of visualizations (Table, Pie, Line, Area, or Bar) by using the **Type** menu buttons in the notebook:



Next steps

- What is the Team Data Science Process?
- Compare the machine learning products and technologies from Microsoft
- Machine learning at scale

Solutions for the automotive, mobility, and transportation industries

3/10/2022 • 2 minutes to read • [Edit Online](#)

The automotive, mobility, and transportation industries work to satisfy the ever-present need to move people and things safely, quickly, and efficiently. Powerful new technologies like cloud computing, IoT, AI, and machine learning can help companies meet that need. Azure provides services to help companies exploit the opportunities and meet the challenges that come with rapidly evolving digital technology.

The automotive industry includes truck and automobile manufacturing and sales, and related parts industries. The design and manufacturing aspects of the industry can take advantage of solutions that address those aspects for many industries, solutions such as [Azure for manufacturing](#).

Another Azure solution, [Azure high-performance computing \(HPC\) for automotive](#), addresses issues that are specific to automotive, such as vehicle engineering, aerodynamic and physics simulations, sensor performance, and autonomous driving software. It offers a wide variety of specialized virtual machines for these areas and many others.

The mobility services industry improves urban mobility with multi-modal route planning, mobile payment and ticketing, vehicle tracking, and analytics for planning and optimization. For related solutions on Azure, see [Emerging mobility services](#).

View the ways that the digital transformation is revolutionizing the automotive and mobility services industry:

Architectures for automotive, mobility, and transportation

The following articles provide detailed analysis of architectures created and recommended for the automotive, mobility, and transportation industries.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Automated guided vehicles fleet control	An example architecture that provides an end-to-end approach for automotive manufacturers that rely on automated guided vehicles (AGVs).	IoT
Building blocks for autonomous-driving simulation environments	An example architecture for automating driving tests, developing control systems, recording vehicle data, and simulating complex driving scenarios.	Containers
Data science and machine learning with Azure Databricks	A solution idea for quick and cost-effective training, deployment, and life-cycle management of thousands of parallel machine learning models.	AI

Architecture	Summary	Technology Focus
Efficient Docker image deployment for intermittent low-bandwidth connectivity	An example architecture for situations where containers are part of the solution and connectivity is intermittent with low bandwidth. Possible uses are over-the-air automotive updates and other mobile scenarios.	IoT
Process real-time vehicle data using IoT	An example architecture for capturing, analyzing, and visualizing data from sensors and IoT devices—key capabilities for creating connected-car solutions.	IoT
Real-time IoT updates	An example architecture that outlines a way for clients like web pages or mobile apps to receive updates from devices in real time without HTTP polling. Instead, Azure SignalR Service pushes content to clients as soon as it's available.	IoT
Run CFD simulations	An example architecture that runs Computational Fluid Dynamics (CFD) simulations. It uses Azure to address the high-compute issues of cost, spare capacity, and long queue times.	Compute

Solution ideas for the automotive, mobility, and transportation industries

The following are other ideas that you can use as a starting point for your energy or environment solution.

- [COVID-19 safe environments with IoT Edge monitoring and alerting](#)
- [Environment monitoring and supply chain optimization with IoT](#)
- [IoT analytics with Azure Data Explorer](#)
- [Machine teaching with the Microsoft Autonomous Systems platform](#)
- [Predictive aircraft engine monitoring](#)
- [Predictive insights with vehicle telematics](#)
- [Real-time asset tracking and management](#)

Solutions for the telecommunications industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

Transform how cloud and edge work together to capitalize on the new service opportunities enabled by 5G. Connect your customers to the right products and services. Use AI, automation, and advanced analytics to realize efficiencies, avoid service disruptions, and drive down costs. To learn more, check out this article, which provides an overview of architectures and ideas for using Azure services to build solutions for the telecommunications industry.

In this [short video](#), Satya Nadella talks about [Azure for operators](#), the Microsoft hybrid cloud solutions that can help service providers deploy, secure, and monetize network services across Azure and other platforms in a multi-cloud environment.

This video describes how Microsoft enables telecommunications organizations to achieve more:

Architectures for telecommunications

The following articles provide detailed analysis of architectures created and recommended for the telecommunications industry.

Architecture	Summary	Technology Focus
Anomaly detector process	Learn about Anomaly Detector and how anomaly detection models are selected for time-series data. You can use time-series data to predict mobile traffic and service needs.	Analytics
Azure Arc hybrid management and deployment for Kubernetes clusters	Learn how Azure Arc extends Kubernetes cluster management and configuration across your datacenters, edge locations, and multiple cloud environments.	Hybrid/Multicloud
Build a speech-to-text transcription pipeline with Azure Cognitive Services	Build an audio ingestion and speech-to-text transcription pipeline by using Cognitive Services. Speech recognition can be used to analyze customer calls.	AI / Machine learning
Customer churn prediction using real-time analytics	Predict customer churn and find patterns in existing data that's associated with the predicted churn rate by using the Microsoft AI platform.	AI / Machine learning
Deploy AI and machine learning at the edge	Learn how to extend rapid machine learning inference from the cloud to on-premises and edge locations by using Azure Stack Edge.	AI / Machine learning

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Determine customer lifetime value and churn prediction with Azure AI services	Learn how to create a solution for predicting customer lifetime value and churn by using Azure Machine Learning. This solution can help you tailor products and services to your best customers.	AI / Machine learning
Enterprise-grade conversational bot	Learn how to build a robust, highly secure, and actively learning chatbot by using the Microsoft Bot Framework.	AI / Machine learning
Predictive maintenance with the intelligent IoT Edge	Learn how you can implement predictive maintenance by using machine learning on the intelligent Azure IoT Edge platform.	IoT
Real-time fraud detection	Analyze data in real time to detect fraudulent transactions or other anomalous activity. This architecture can be used to detect fraudulent mobile-phone calls.	Security

Solution ideas for telecommunications

The following are other ideas that you can use as a starting point for your telecommunications solution.

- [Content Delivery Network analytics](#)
- [IoT-connected light, power, and internet for emerging markets](#)
- [IoT device connectivity for healthcare facilities](#)
- [IoT Edge data storage and processing](#)
- [Low-latency network connections for industry](#)
- [Speech services](#)
- [Video capture and analytics](#)

Solutions for the facilities and real estate industries

3/10/2022 • 3 minutes to read • [Edit Online](#)

IoT, AI, machine learning, and other Azure technologies can help building owners, operators, and occupants save money and live better lives.

Smart buildings can heat and cool themselves, sending energy to the right places at the right times. They can even greet visitors and help them find their way. Predictive maintenance can help building operators know when something is likely to need attention.

Machine learning, Azure IoT Edge, and other technologies can help create safe indoor environments during challenging times. And solar-powered IoT devices combined with Azure services can provide clean, low-cost power, light, and internet service to remote customers.

Watch this short video to get a sense of some of the things smart buildings can do:

Architectures for facilities and real estate

The following articles provide detailed analysis of architectures created and recommended for the facilities and real-estate industries.

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Azure IoT reference architecture	Learn about a recommended architecture for IoT applications that use Azure platform as a service (PaaS) components. Apply this architecture to smart buildings and other facilities scenarios.	IoT
Buy online, pick up in store (BOPIS)	Learn how to use Azure IoT to enhance a solution for stores that implement buy online, pick up in store solutions.	IoT
Cognizant Safe Buildings with IoT and Azure	Deploy Cognizant Safe Buildings, IoT, and Azure services to help protect buildings from COVID-19 outbreaks.	IoT
Connected factory signal pipeline	Learn about the connected factory signal pipeline architecture, which provides a common configuration interface to connect brownfield IoT devices through an OPC Unified Architecture (UA) gateway. Apply this architecture to predictive maintenance scenarios and other scenarios that are related to facilities and real estate.	IoT

Architecture	Summary	Technology Focus
COVID-19 safe solutions with IoT Edge	Create an environment that monitors social distance, mask/PPE use, and occupancy requirements by using CCTVs and Azure IoT Edge, Azure Stream Analytics, and Azure Machine Learning.	Containers
Create smart places by using Azure Digital Twins	Use Azure Digital Twins to create models of smart places from IoT device data. View and control products, systems, environments, and experiences.	IoT
Digital image-based modeling on Azure	Learn how to perform image-based modeling on Azure infrastructure as a service (IaaS). You can use this technology to model and measure buildings.	Compute
Facilities management powered by mixed reality and IoT	Improve uptime and operations with mixed reality and IoT. Visualize a virtual replica of your physical space with real-time data in the context of your environment.	Mixed reality
IoT connected light, power, and internet for emerging markets	View an architecture that uses solar-powered IoT devices with Azure services to provide clean, low-cost power, light, and internet service to remote customers.	IoT
IoT device connectivity for healthcare facilities	Learn how to reliably connect building and campus IoT devices to the cloud with improved security and scalability.	Networking
IoT measure and control loops	Learn how an IoT measure and control loop keeps an IoT device within the tolerable range of setpoint configuration. This technology applies to smoke sensors, thermostats, and solar panels.	IoT
IoT monitor and manage loops	Learn about an IoT monitor and manage loop—a supervisory system that continually monitors a physical system that's controlled by a set of networked IoT devices. This technology applies to smart campuses and smart metering.	IoT
UVEN smart and secure disinfection and lighting	Learn about a system that uses IoT and Azure Sphere to provide smart virus disinfection and healthy, human-optimized lighting.	IoT

Solution ideas for facilities and real estate

The following are other ideas that you can use as a starting point for your facilities and real estate solution.

- [Azure digital twins builder](#)
- [Contactless IoT interfaces with Azure intelligent edge](#)
- [Control IoT devices using a voice assistant](#)
- [Environment monitoring and supply chain optimization with IoT](#)
- [IoT Connected Platform for COVID-19 detection and prevention](#)
- [IoT monitor and manage loops](#)
- [Low-latency network connections for industry](#)
- [Predict length of stay and patient flow](#)

Solutions for the education industry

3/10/2022 • 2 minutes to read • [Edit Online](#)

Help your students take hold of their futures with Microsoft Azure, a platform that provides the tools to enable, elevate, and enhance remote learning. Azure services can connect teachers to students, help you create models of smart campus buildings, deploy virtual labs, and more.

Watch the following short video to get a sense of the breadth of the Azure offerings for education. It also shows how the [Cloud Adoption Framework](#) can help you get the necessary resources for education.

Architectures for education

The following articles provide detailed analysis of architectures created and recommended for the education industry.

Architecture	Summary	Technology Focus
Build a telehealth system on Azure	Learn how to use Azure cloud services to build a system that connects teachers to remote students.	Containers
Computer forensics chain of custody in Azure	Create an infrastructure to ensure a valid digital-evidence chain of custody for computer forensics in Azure.	Management
Create smart places by using Azure Digital Twins	Use Azure Digital Twins to create models of smart buildings and campuses from IoT device data. View and control systems, environments, and experiences.	IoT
Enhanced-security hybrid messaging — client access	Learn how to enhance the security of your educational messaging infrastructure in a client access scenario by using Azure AD Multi-Factor Authentication.	Hybrid
Enhanced-security hybrid messaging — mobile access	Learn how to enhance the security for your educational messaging infrastructure in a mobile access scenario by using Azure AD Multi-Factor Authentication.	Hybrid
Enhanced-security hybrid messaging — web access	Learn how to enhance the security of your educational messaging infrastructure in a web access scenario by using Azure AD Multi-Factor Authentication.	Hybrid

ARCHITECTURE	SUMMARY	TECHNOLOGY FOCUS
Governance of Teams guest users	Learn how to use Microsoft Teams and Azure AD entitlement management to collaborate, while maintaining control over resource use.	Identity
Moodle deployment with Azure NetApp Files	Deploy the Moodle learning platform with Azure NetApp Files for a resilient solution that offers high-throughput, low-latency access to scalable shared storage.	Storage
Optimized storage with logical data classification	Learn about a high-availability solution that handles massive amounts of data. This solution applies to scenarios like university enrollment and scheduling.	Databases
Secure research environment for regulated data	Learn about an architecture created for higher education research institutions. It enables researchers to access sensitive data while providing a high level of control and data protection.	Security
Teacher-provisioned virtual labs in Azure	Use Azure Lab Services to set up identical virtual labs from templates. Virtual labs are ideal for training and other scenarios.	DevOps

Solution ideas for education

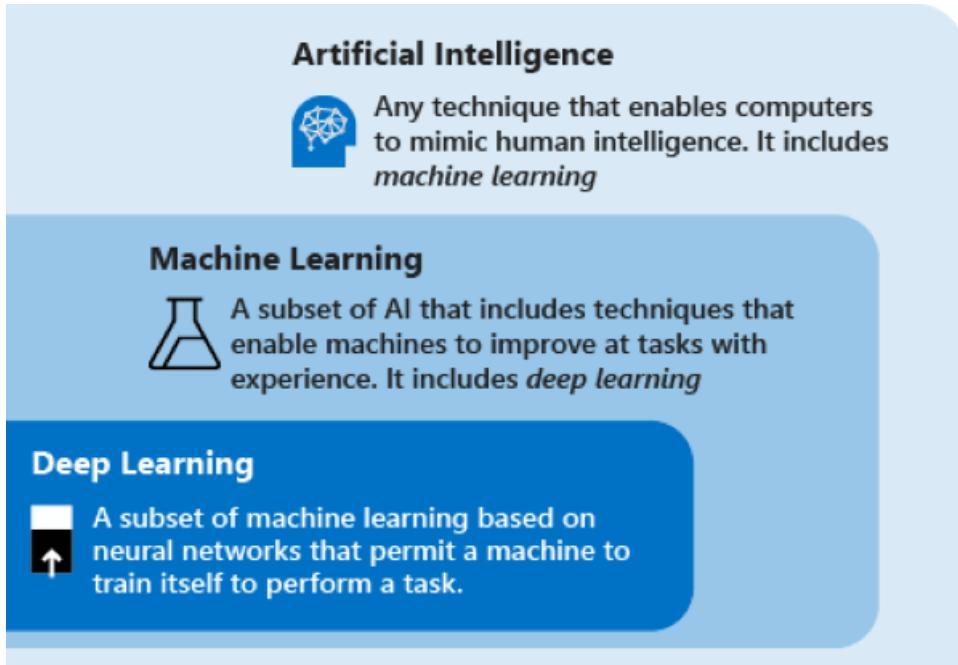
The following are other ideas that you can use as a starting point for your education solution.

- [COVID-19 safe solutions with IoT Edge](#)
- [IoT connected light, power, and internet for emerging markets](#)
- [Project 15 Open Platform IoT sustainability](#)

Artificial intelligence (AI) architecture design

3/10/2022 • 15 minutes to read • [Edit Online](#)

Artificial intelligence (AI) is the capability of a computer to imitate intelligent human behavior. Through AI, machines can analyze images, comprehend speech, interact in natural ways, and make predictions using data.



AI concepts

Algorithm

An *algorithm* is a sequence of calculations and rules used to solve a problem or analyze a set of data. It is like a flow chart, with step-by-step instructions for questions to ask, but written in math and programming code. An algorithm may describe how to determine whether a pet is a cat, dog, fish, bird, or lizard. Another far more complicated algorithm may describe how to identify a written or spoken language, analyze its words, translate them into a different language, and then check the translation for accuracy.

Machine learning

Machine learning (ML) is an AI technique that uses mathematical algorithms to create predictive models. An algorithm is used to parse data fields and to "learn" from that data by using patterns found within it to generate models. Those models are then used to make informed predictions or decisions about new data.

The predictive models are validated against known data, measured by performance metrics selected for specific business scenarios, and then adjusted as needed. This process of learning and validation is called *training*. Through periodic retraining, ML models are improved over time.

- [Machine learning at scale](#)
- [What are the machine learning products at Microsoft?](#)

Deep learning

Deep learning is a type of ML that can determine for itself whether its predictions are accurate. It also uses algorithms to analyze data, but it does so on a larger scale than ML.

Deep learning uses artificial neural networks, which consist of multiple layers of algorithms. Each layer looks at

the incoming data, performs its own specialized analysis, and produces an output that other layers can understand. This output is then passed to the next layer, where a different algorithm does its own analysis, and so on.

With many layers in each neural network-and sometimes using multiple neural networks-a machine can learn through its own data processing. This requires much more data and much more computing power than ML.

- [Deep learning versus machine learning](#)
- [Distributed training of deep learning models on Azure](#)
- [Batch scoring of deep learning models on Azure](#)
- [Training of Python scikit-learn and deep learning models on Azure](#)
- [Real-time scoring of Python scikit-learn and deep learning models on Azure](#)

Bots

A *bot* is an automated software program designed to perform a particular task. Think of it as a robot without a body. Early bots were comparatively simple, handling repetitive and voluminous tasks with relatively straightforward algorithmic logic. An example would be web crawlers used by search engines to automatically explore and catalog web content.

Bots have become much more sophisticated, using AI and other technologies to mimic human activity and decision-making, often while interacting directly with humans through text or even speech. Examples include bots that can take a dinner reservation, chatbots (or conversational AI) that help with customer service interactions, and social bots that post breaking news or scientific data to social media sites.

Microsoft offers the Azure Bot Service, a managed service purpose-built for enterprise-grade bot development.

- [About Azure Bot Service](#)
- [Ten guidelines for responsible bots](#)
- [Azure reference architecture: Enterprise-grade conversational bot](#)
- [Example workload: Conversational chatbot for hotel reservations on Azure](#)

Autonomous systems

Autonomous systems are part of an evolving new class that goes beyond basic automation. Instead of performing a specific task repeatedly with little or no variation (like bots do), autonomous systems bring intelligence to machines so they can adapt to changing environments to accomplish a desired goal.

Smart buildings use autonomous systems to automatically control operations like lighting, ventilation, air conditioning, and security. A more sophisticated example would be a self-directed robot exploring a collapsed mine shaft to thoroughly map its interior, determine which portions are structurally sound, analyze the air for breathability, and detect signs of trapped miners in need of rescue-all without a human monitoring in real time on the remote end.

- [Autonomous systems and solutions from Microsoft AI](#)

General info on Microsoft AI

Learn more about Microsoft AI, and keep up-to-date with related news:

- [Microsoft AI School](#)
- [Azure AI platform page](#)
- [Microsoft AI platform page](#)
- [Microsoft AI Blog](#)

- [Microsoft AI on GitHub: Samples, reference architectures, and best practices](#)
- [Azure Architecture Center](#)

High-level architectural types

Prebuilt AI

Prebuilt AI is exactly what it sounds like—off-the-shelf AI models, services, and APIs that are ready to use. These help you add intelligence to apps, websites, and flows without having to gather data and then build, train, and publish your own models.

One example of prebuilt AI might be a pretrained model that can be incorporated as is or used to provide a baseline for further custom training. Another example would be a cloud-based API service that can be called at will to process natural language in a desired fashion.

Azure Cognitive Services

[Cognitive Services](#) provide developers the opportunity to use prebuilt APIs and integration toolkits to create applications that can see, hear, speak, understand, and even begin to reason. The catalog of services within Cognitive Services can be categorized into five main pillars: Vision, Speech, Language, Web Search, and Decision/Recommendation.

- [Azure Cognitive Services documentation](#)
- [Try Azure Cognitive Services for free](#)
- [Choosing a Microsoft Cognitive Services technology](#)
- [Choosing a natural language processing technology in Azure](#)

Prebuilt AI models in AI Builder

AI Builder is a new capability in [Microsoft Power Platform](#) that provides a point-and-click interface for adding AI to your apps, even if you have no coding or data science skills. (Some features in AI Builder have not yet released for general availability and remain in preview status. For more information, refer to the [Feature availability by region](#) page.)

You can build and train your own models, but AI Builder also provides [select prebuilt AI models](#) that are ready for use right away. For example, you can add a component in Microsoft Power Apps based on a prebuilt model that recognizes contact information from business cards.

- [Power Apps on Azure](#)
- [AI Builder documentation](#)
- [AI model types in AI Builder](#)
- [Overview of prebuilt AI models in AI Builder](#)

Custom AI

Although prebuilt AI is useful (and increasingly flexible), the best way to get what you need from AI is probably to build a system yourself. This is obviously a very deep and complex subject, but let's look at some basic concepts beyond what we've just covered.

Code languages

The core concept of AI is the use of algorithms to analyze data and generate models to describe (or *score*) it in ways that are useful. Algorithms are written by developers and data scientists (and sometimes by other algorithms) using programming code. Two of the most popular programming languages for AI development are currently Python and R.

[Python](#) is a general-purpose, high-level programming language. It has a simple, easy-to-learn syntax that

emphasizes readability. There is no compiling step. Python has a large standard library, but it also supports the ability to add modules and packages. This encourages modularity and lets you expand capabilities when needed. There is a large and growing ecosystem of AI and ML libraries for Python, including many that are readily available in Azure.

- [Python on Azure product home page](#)
- [Azure for Python developers](#)
- [Azure Machine Learning SDK for Python](#)
- [Introduction to machine learning with Python and Azure Notebooks](#)
- [scikit-learn](#). An open-source ML library for Python
- [PyTorch](#). An open-source Python library with a rich ecosystem that can be used for deep learning, computer vision, natural language processing, and more
- [TensorFlow](#). An open-source symbolic math library also used for ML applications and neural networks
- [Tutorial: Apply machine learning models in Azure Functions with Python and TensorFlow](#)

R is a [language and environment](#) for statistical computing and graphics. It can be used for everything from mapping broad social and marketing trends online to developing financial and climate models.

Microsoft has fully embraced the R programming language and provides many different options for R developers to run their code in Azure.

- [R developer's guide to Azure](#)
- [Microsoft R Open](#). An enhanced distribution of R from Microsoft, fully compatible with R-3.5.3, with additional capabilities for improved performance and reproducibility, in addition to support for Windows- and Linux-based platforms
- [Tutorial: Create a logistic regression model in R with Azure Machine Learning](#)

Training

Training is core to machine learning. It is the iterative process of "teaching" an algorithm to create models, which are used to analyze data and then make accurate predictions from it. In practice, this process has three general phases: training, validation, and testing.

During the training phase, a quality set of known data is tagged so that individual fields are identifiable. The tagged data is fed to an algorithm configured to make a particular prediction. When finished, the algorithm outputs a model that describes the patterns it found as a set of parameters. During validation, fresh data is tagged and used to test the model. The algorithm is adjusted as needed and possibly put through more training. Finally, the testing phase uses real-world data without any tags or preselected targets. Assuming the model's results are accurate, it is considered ready for use and can be deployed.

- [Train models with Azure Machine Learning](#)

Hyperparameter tuning

Hyperparameters are data variables that govern the training process itself. They are configuration variables that control how the algorithm operates. Hyperparameters are thus typically set before model training begins and are not modified within the training process in the way that parameters are. Hyperparameter tuning involves running trials within the training task, assessing how well they are getting the job done, and then adjusting as needed. This process generates multiple models, each trained using different families of hyperparameters.

- [Tune hyperparameters for your model with Azure Machine Learning](#)

Model selection

The process of training and hyperparameter tuning produces numerous candidate models. These can have

many different variances, including the effort needed to prepare the data, the flexibility of the model, the amount of processing time, and of course the degree of accuracy of its results. Choosing the best trained model for your needs and constraints is called *model selection*, but this is as much about preplanning before training as it is about choosing the one that works best.

Automated machine learning (AutoML)

Automated machine learning, also known as AutoML, is the process of automating the time-consuming, iterative tasks of machine learning model development. It can significantly reduce the time it takes to get production-ready ML models. Automated ML can assist with model selection, hyperparameter tuning, model training, and other tasks, without requiring extensive programming or domain knowledge.

- [What is automated machine learning?](#)

Scoring

Scoring is also called *prediction* and is the process of generating values based on a trained machine learning model, given some new input data. The values, or scores, that are created can represent predictions of future values, but they might also represent a likely category or outcome. The scoring process can generate many different types of values:

- A list of recommended items and a similarity score
- Numeric values, for time series models and regression models
- A probability value, indicating the likelihood that a new input belongs to some existing category
- The name of a category or cluster to which a new item is most similar
- A predicted class or outcome, for classification models

Batch scoring is when data is collected during some fixed period of time and then processed in a batch. This might include generating business reports or analyzing customer loyalty.

Real-time scoring is exactly that—scoring that is ongoing and performed as quickly as possible. The classic example is credit card fraud detection, but real-time scoring can also be used in speech recognition, medical diagnoses, market analyses, and many other applications.

General info on custom AI on Azure

- [Microsoft AI on GitHub: Samples, reference architectures, and best practices](#)
- [Custom AI on Azure GitHub repo](#). A collection of scripts and tutorials to help developers effectively use Azure for their AI workloads
- [Azure Machine Learning SDK for Python](#)
- [Azure Machine Learning service example notebooks \(Python\)](#). A GitHub repo of example notebooks demonstrating the Azure Machine Learning Python SDK
- [Azure Machine Learning SDK for R](#)

Azure AI platform offerings

Following is a breakdown of Azure technologies, platforms, and services you can use to develop AI solutions for your needs.

Azure Machine Learning

This is an enterprise-grade machine learning service to build and deploy models faster. Azure Machine Learning offers web interfaces and SDKs so you can quickly train and deploy your machine learning models and pipelines at scale. Use these capabilities with open-source Python frameworks, such as PyTorch, TensorFlow, and scikit-learn.

- [What are the machine learning products at Microsoft?](#)
- [Azure Machine Learning product home page](#)
- [Azure Machine Learning Data Architecture Guide overview](#)
- [Azure Machine Learning documentation overview](#)
- [What is Azure Machine Learning?](#) General orientation with links to many learning resources, SDKs, documentation, and more

Machine learning reference architectures for Azure

- [Training of Python scikit-learn and deep learning models on Azure](#)
- [Distributed training of deep learning models on Azure](#)
- [Batch scoring of Python machine learning models on Azure](#)
- [Batch scoring of deep learning models on Azure](#)
- [Real-time scoring of Python scikit-learn and deep learning models on Azure](#)
- [Machine learning operationalization \(MLOps\) for Python models using Azure Machine Learning](#)
- [Batch scoring of R machine learning models on Azure](#)
- [Real-time scoring of R machine learning models on Azure](#)
- [Batch scoring of Spark machine learning models on Azure Databricks](#)
- [Enterprise-grade conversational bot](#)
- [Build a real-time recommendation API on Azure](#)

Azure automated machine learning

Azure provides extensive support for automated ML. Developers can build models using a no-code UI or through a code-first notebooks experience.

- [Azure automated machine learning product home page](#)
- [Azure automated ML infographic \(PDF\)](#)
- [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#)
- [Tutorial: Use automated machine learning to predict taxi fares](#)
- [Configure automated ML experiments in Python](#)
- [Use the CLI extension for Azure Machine Learning](#)
- [Automate machine learning activities with the Azure Machine Learning CLI](#)

Azure Cognitive Services

This is a comprehensive family of AI services and cognitive APIs to help you build intelligent apps. These domain-specific, pretrained AI models can be customized with your data.

- [Cognitive Services product home page](#)
- [Azure Cognitive Services documentation](#)

Azure Cognitive Search

This is an AI-powered cloud search service for mobile and web app development. The service can search over private heterogenous content, with options for AI enrichment if your content is unstructured or unsearchable in

raw form.

- [Azure Cognitive Search product home page](#)
- [Getting started with AI enrichment](#)
- [Azure Cognitive Search documentation overview](#)
- [Choosing a natural language processing technology in Azure](#)
- [Quickstart: Create an Azure Cognitive Search cognitive skill set in the Azure portal](#)

Azure Bot Service

This is a purpose-built bot development environment with out-of-the-box templates to get started quickly.

- [Azure Bot Service product home page](#)
- [Azure Bot Service documentation overview](#)
- [Azure reference architecture: Enterprise-grade conversational bot](#)
- [Example workload: Conversational chatbot for hotel reservations on Azure](#)
- [Microsoft Bot Framework](#)
- [GitHub Bot Builder repo](#)

Apache Spark on Azure

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big data analytic applications. Spark provides primitives for in-memory cluster computing. A Spark job can load and cache data into memory and query it repeatedly, which is much faster than disk-based applications, such as Hadoop.

[Apache Spark in Azure HDInsight](#) is the Microsoft implementation of Apache Spark in the cloud. Spark clusters in HDInsight are compatible with Azure Storage and Azure Data Lake Storage, so you can use HDInsight Spark clusters to process your data stored in Azure.

The Microsoft Machine Learning library for Apache Spark is [MMLSpark](#) (Microsoft ML for Apache Spark). It is an open-source library that adds many deep learning and data science tools, networking capabilities, and production-grade performance to the Spark ecosystem. [Learn more about MMLSpark features and capabilities](#).

- [Azure HDInsight overview](#). Basic information about features, cluster architecture, and use cases, with pointers to quickstarts and tutorials.
- [Tutorial: Build an Apache Spark machine learning application in Azure HDInsight](#)
- [Apache Spark best practices on HDInsight](#)
- [Configure HDInsight Apache Spark Cluster settings](#)
- [Machine learning on HDInsight](#)
- [GitHub repo for MMLSpark: Microsoft Machine Learning library for Apache Spark](#)
- [Create an Apache Spark machine learning pipeline on HDInsight](#)

Azure Databricks Runtime for Machine Learning

[Azure Databricks](#) is an Apache Spark–based analytics platform with one-click setup, streamlined workflows, and an interactive workspace for collaboration between data scientists, engineers, and business analysts.

[Databricks Runtime for Machine Learning \(Databricks Runtime ML\)](#) lets you start a Databricks cluster with all of the libraries required for distributed training. It provides a ready-to-go environment for machine learning and

data science. Plus, it contains multiple popular libraries, including TensorFlow, PyTorch, Keras, and XGBoost. It also supports distributed training using Horovod.

- [Azure Databricks product home page](#)
- [Azure Databricks documentation](#)
- [Machine learning capabilities in Azure Databricks](#)
- [How-to guide: Databricks Runtime for Machine Learning](#)
- [Batch scoring of Spark machine learning models on Azure Databricks](#)
- [Deep learning overview for Azure Databricks](#)

Customer stories

Different industries are applying AI in innovative and inspiring ways. Following are a number of customer case studies and success stories:

- [ASOS: Online retailer solves challenges with Azure Machine Learning service](#)
- [KPMG helps financial institutions save millions in compliance costs with Azure Cognitive Services](#)
- [Volkswagen: Machine translation speaks Volkswagen – in 40 languages](#)
- [Buncee: NYC school empowers readers of all ages and abilities with Azure AI](#)
- [InterSystems: Data platform company boosts healthcare IT by generating critical information at unprecedented speed](#)
- [Zencity: Data-driven startup uses funding to help local governments support better quality of life for residents](#)
- [Bosch uses IoT innovation to drive traffic safety improvements by helping drivers avoid serious accidents](#)
- [Automation Anywhere: Robotic process automation platform developer enriches its software with Azure Cognitive Services](#)
- [Wix deploys smart, scalable search across 150 million websites with Azure Cognitive Search](#)
- [Asklepios Klinik Altona: Precision surgeries with Microsoft HoloLens 2 and 3D visualization](#)
- [AXA Global P&C: Global insurance firm models complex natural disasters with cloud-based HPC](#)

[Browse more AI customer stories](#)

Next steps

- To learn about the artificial intelligence development products available from Microsoft, refer to the [Microsoft AI platform](#) page.
- For training in how to develop AI solutions, refer to [Microsoft AI School](#).
- [Microsoft AI on GitHub: Samples, reference architectures, and best practices](#) organizes the Microsoft open source AI-based repositories, providing tutorials and learning materials.

Choose a Microsoft cognitive services technology

3/10/2022 • 3 minutes to read • [Edit Online](#)

Microsoft cognitive services are cloud-based APIs that you can use in artificial intelligence (AI) applications and data flows. They provide you with pretrained models that are ready to use in your application, requiring no data and no model training on your part. The cognitive services are developed by Microsoft's AI and Research team and leverage the latest deep learning algorithms. They are consumed over HTTP REST interfaces. In addition, SDKs are available for many common application development frameworks.

The cognitive services include:

- Text analysis
- Computer vision
- Video analytics
- Speech recognition and generation
- Natural language understanding
- Intelligent search

Key benefits:

- Minimal development effort for state-of-the-art AI services.
- Easy integration into apps via HTTP REST interfaces.
- Built-in support for consuming cognitive services in Azure Data Lake Analytics.

Considerations:

- Only available over the web. Internet connectivity is generally required. An exception is the Custom Vision Service, whose trained model you can export for prediction on devices and at the IoT edge.
- Although considerable customization is supported, the available services may not suit all predictive analytics requirements.

What are your options when choosing amongst the cognitive services?

In Azure, there are dozens of Cognitive Services available. The current listing of these is available in a directory categorized by the functional area they support:

- [Vision](#)
- [Speech](#)
- [Decision](#)
- [Search](#)
- [Language](#)

Key selection criteria

To narrow the choices, start by answering these questions:

- What type of data are you dealing with? Narrow your options based on the type of input data you are working with. For example, if your input is text, select from the services that have an input type of text.

- Do you have the data to train a model? If yes, consider the custom services that enable you to train their underlying models with data that you provide, for improved accuracy and performance.

Capability matrix

The following tables summarize the key differences in capabilities.

Uses prebuilt models

CAPABILITY	INPUT TYPE	KEY BENEFIT
Text Analytics API	Text	Evaluate sentiment and topics to understand what users want.
Entity Linking API	Text	Power your app's data links with named entity recognition and disambiguation.
Language Understanding Intelligent Service (LUIS)	Text	Teach your apps to understand commands from your users.
QnA Maker Service	Text	Distill FAQ formatted information into conversational, easy-to-navigate answers.
Linguistic Analysis API	Text	Simplify complex language concepts and parse text.
Knowledge Exploration Service	Text	Enable interactive search experiences over structured data via natural language inputs.
Web Language Model API	Text	Use predictive language models trained on web-scale data.
Academic Knowledge API	Text	Tap into the wealth of academic content in the Microsoft Academic Graph populated by Bing.
Bing Autosuggest API	Text	Give your app intelligent autosuggest options for searches.
Bing Spell Check API	Text	Detect and correct spelling mistakes in your app.
Translator Text API	Text	Machine translation.
Recommendations API	Text	Predict and recommend items your customers want.
Bing Entity Search API	Text (web search query)	Identify and augment entity information from the web.
Bing Image Search API	Text (web search query)	Search for images.
Bing News Search API	Text (web search query)	Search for news.

CAPABILITY	INPUT TYPE	KEY BENEFIT
Bing Video Search API	Text (web search query)	Search for videos.
Bing Web Search API	Text (web search query)	Get enhanced search details from billions of web documents.
Bing Speech API	Text or Speech	Convert speech to text and back again.
Speaker Recognition API	Speech	Use speech to identify and authenticate individual speakers.
Translator Speech API	Speech	Perform real-time speech translation.
Computer Vision API	Images (or frames from video)	Distill actionable information from images, automatically create description of photos, derive tags, recognize celebrities, extract text, and create accurate thumbnails.
Content Moderator	Text, Images or Video	Automated image, text, and video moderation.
Emotion API	Images (photos with human subjects)	Identify the range emotions of human subjects.
Face API	Images (photos with human subjects)	Detect, identify, analyze, organize, and tag faces in photos.
Video Indexer	Video	Video insights such as sentiment, transcript speech, translate speech, recognize faces and emotions, and extract keywords.

Trained with custom data you provide

CAPABILITY	INPUT TYPE	KEY BENEFIT
Custom Vision Service	Images (or frames from video)	Customize your own computer vision models.
Custom Speech Service	Speech	Overcome speech recognition barriers like speaking style, background noise, and vocabulary.
Custom Decision Service	Web content (for example, RSS feed)	Use machine learning to automatically select the appropriate content for your home page
Bing Custom Search API	Text (web search query)	Commercial-grade search tool.

R developer's guide to Azure

3/10/2022 • 6 minutes to read • [Edit Online](#)

Many data scientists dealing with ever-increasing volumes of data are looking for ways to harness the power of cloud computing for their analyses. This article provides an overview of the various ways that data scientists can use their existing skills with the [R programming language](#) in Azure.

Microsoft has fully embraced the R programming language as a first-class tool for data scientists. By providing many different options for R developers to run their code in Azure, the company is enabling data scientists to extend their data science workloads into the cloud when tackling large-scale projects.

Let's examine the various options and the most compelling scenarios for each one.



Azure services with R language support

This article covers the following Azure services that support the R language:

SERVICE	DESCRIPTION
Data Science Virtual Machine	a customized VM to use as a data science workstation or as a custom compute target
ML Services on HDInsight	cluster-based system for running R analyses on large datasets across many nodes
Azure Databricks	collaborative Spark environment that supports R and other languages
Azure Machine Learning	cloud service that you use to train, deploy, automate, and manage machine learning models
Azure Batch	offers a variety options for economically running R code across many nodes in a cluster
Azure SQL Managed Instance	run R and Python scripts inside of the SQL Server database engine

Data Science Virtual Machine

The [Data Science Virtual Machine](#) (DSVM) is a customized VM image on Microsoft's Azure cloud platform built specifically for doing data science. It has many popular data science tools, including:

- [Microsoft R Open](#)
- [RStudio Desktop](#)
- [RStudio Server](#)

The DSVM can be provisioned with either Windows or Linux as the operating system. You can use the DSVM in two different ways: as an interactive workstation or as a compute platform for a custom cluster.

As a workstation

If you want to get started with R in the cloud quickly and easily, this is your best bet. The environment will be familiar to anyone who has worked with R on a local workstation. However, instead of using local resources, the R environment runs on a VM in the cloud. If your data is already stored in Azure, this has the added benefit of allowing your R scripts to run "closer to the data." Instead of transferring the data across the Internet, the data can be accessed over Azure's internal network, which provides much faster access times.

The DSVM can be particularly useful to small teams of R developers. Instead of investing in powerful workstations for each developer and requiring team members to synchronize on which versions of the various software packages they will use, each developer can spin up an instance of the DSVM whenever needed.

As a compute platform

In addition to being used as a workstation, the DSVM is also used as an elastically scalable compute platform for R projects. Using the [AzureDSVM](#) R package, you can programmatically control the creation and deletion of DSVM instances. You can form the instances into a cluster and deploy a distributed analysis to be performed in the cloud. This entire process can be controlled by R code running on your local workstation.

To learn more about the DSVM, see [Introduction to Azure Data Science Virtual Machine for Linux and Windows](#).

ML Services on HDInsight

[Microsoft ML Services](#) provide data scientists, statisticians, and R programmers with on-demand access to scalable, distributed methods of analytics on HDInsight. This solution provides the latest capabilities for R-based analytics on datasets of virtually any size, loaded to either Azure Blob or Data Lake storage.

This is an enterprise-grade solution that allows you to scale your R code across a cluster. By using functions in Microsoft's [RevoScaler](#) package, your R scripts on HDInsight can run data processing functions in parallel across many nodes in a cluster. This allows R to crunch data on a much larger scale than is possible with single-threaded R running on a workstation.

This ability to scale makes ML Services on HDInsight a great option for R developers with massive data sets. It provides a flexible and scalable platform for running your R scripts in the cloud.

For a walk-through on creating an ML Services cluster, see [Get started with ML Services on Azure HDInsight](#).

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform. Designed with the founders of Apache Spark, Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts.

The collaboration in Databricks is enabled by the platform's notebook system. Users can create, share, and edit notebooks with other users of the systems. These notebooks allow users to write code that executes against Spark clusters managed in the Databricks environment. These notebooks fully support R and give users access to Spark through both the [SparkR](#) and [sparklyr](#) packages.

Since Databricks is built on Spark and has a strong focus on collaboration, the platform is often used by teams of data scientists that work together on complex analyses of large data sets. Because the notebooks in Databricks support other languages in addition to R, it is especially useful for teams where analysts use different languages for their primary work.

The article [What is Azure Databricks?](#) can provide more details about the platform and help you get started.

Azure Machine Learning

[Azure Machine Learning](#) can be used for any kind of machine learning, from classical machine learning to deep

learning, supervised and unsupervised learning. Whether you prefer to write Python or R code or zero-code/low-code options such as the designer, you can build, train and track highly accurate machine learning and deep-learning models in an Azure Machine Learning Workspace.

Start training on your local machine and then scale out to the cloud. [Train your first model in R](#) with Azure Machine Learning today.

Azure Batch

For large-scale R jobs, you can use [Azure Batch](#). This service provides cloud-scale job scheduling and compute management so you can scale your R workload across tens, hundreds, or thousands of virtual machines. Since it is a generalized computing platform, there are a few options for running R jobs on Azure Batch.

One option for running an R script in Azure Batch is to bundle your code with "RScript.exe" as a Batch App in the Azure portal. For a detailed walkthrough, see [R Workloads on Azure Batch](#).

Another option is to use the [Azure Distributed Data Engineering Toolkit](#) (AZTK), which allows you to provision on-demand Spark clusters using Docker containers in Azure Batch. This provides an economical way to run Spark jobs in Azure. By using [SparklyR with AZTK](#), your R scripts can be scaled out in the cloud easily and economically.

Azure SQL Managed Instance

[Azure SQL Managed Instance](#) is Microsoft's intelligent, scalable, cloud database service. It allows you to use the full power of SQL Server without any hassle of setting up the infrastructure. This includes [Machine Learning Services](#) which contains Microsoft R and Python packages for high-performance predictive analytics and machine learning.

Machine Learning Services offers an embedded, predictive analytics and data science engine that can execute R/Python code within a SQL Server database. Instead of extracting data from the database and loading it into the R/Python environment, you load your R/Python code directly into the database and let it run right alongside the data. The relational data can be used in stored procedures, as T-SQL scripts containing R/Python statements, or as R/Python code containing T-SQL.

While Machine Learning Services has been part of on-premises SQL Server since 2016, it is relatively new to Azure SQL Managed Instance.

Next steps

- [Running your R code on Azure with mrsdeploy](#)
- [R on Azure](#): an overview of packages, tools, and case studies for using R with Azure

Compare the machine learning products and technologies from Microsoft

3/10/2022 • 8 minutes to read • [Edit Online](#)

Learn about the machine learning products and technologies from Microsoft. Compare options to help you choose how to most effectively build, deploy, and manage your machine learning solutions.

Cloud-based machine learning products

The following options are available for machine learning in the Azure cloud.

CLOUD OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Azure Machine Learning	Managed platform for machine learning	Use a pretrained model. Or, train, deploy, and manage models on Azure using Python and CLI
Azure Cognitive Services	Pre-built AI capabilities implemented through REST APIs and SDKs	Build intelligent applications quickly using standard programming languages. Doesn't require machine learning and data science expertise
Azure SQL Managed Instance Machine Learning Services	In-database machine learning for SQL	Train and deploy models inside Azure SQL Managed Instance
Machine learning in Azure Synapse Analytics	Analytics service with machine learning	Train and deploy models inside Azure Synapse Analytics
Machine learning and AI with ONNX in Azure SQL Edge	Machine learning in SQL on IoT	Train and deploy models inside Azure SQL Edge
Azure Databricks	Apache Spark-based analytics platform	Build and deploy models and data workflows using integrations with open-source machine learning libraries and the MLFlow platform.

On-premises machine learning products

The following options are available for machine learning on-premises. On-premises servers can also run in a virtual machine in the cloud.

ON-PREMISES OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
SQL Server Machine Learning Services	In-database machine learning for SQL	Train and deploy models inside SQL Server
Machine Learning Services on SQL Server Big Data Clusters	Machine learning in Big Data Clusters	Train and deploy models on SQL Server Big Data Clusters

Development platforms and tools

The following development platforms and tools are available for machine learning.

PLATFORMS/TOOLS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Azure Data Science Virtual Machine	Virtual machine with pre-installed data science tools	Develop machine learning solutions in a pre-configured environment
ML.NET	Open-source, cross-platform machine learning SDK	Develop machine learning solutions for .NET applications
Windows ML	Windows 10 machine learning platform	Evaluate trained models on a Windows 10 device
MMLSpark	Open-source, distributed, machine learning and microservices framework for Apache Spark	Create and deploy scalable machine learning applications for Scala and Python.
Machine Learning extension for Azure Data Studio	Open-source and cross-platform machine learning extension for Azure Data Studio	Manage packages, import machine learning models, make predictions, and create notebooks to run experiments for your SQL databases

Azure Machine Learning

[Azure Machine Learning](#) is a fully managed cloud service used to train, deploy, and manage machine learning models at scale. It fully supports open-source technologies, so you can use tens of thousands of open-source Python packages such as TensorFlow, PyTorch, and scikit-learn. Rich tools are also available, such as [Compute instances](#), [Jupyter notebooks](#), or the [Azure Machine Learning for Visual Studio Code extension](#), a free extension that allows you to manage your resources, model training workflows and deployments in Visual Studio Code. Azure Machine Learning includes features that automate model generation and tuning with ease, efficiency, and accuracy.

Use Python SDK, Jupyter notebooks, R, and the CLI for machine learning at cloud scale. For a low-code or no-code option, use Azure Machine Learning's interactive [designer](#) in the studio to easily and quickly build, test, and deploy models using pre-built machine learning algorithms.

[Try Azure Machine Learning for free.](#)

Type	Cloud-based machine learning solution
Supported languages	Python, R
Machine learning phases	Model training Deployment MLOps/Management
Key benefits	Code first (SDK) and studio & drag-and-drop designer web interface authoring options. Central management of scripts and run history, making it easy to compare model versions. Easy deployment and management of models to the cloud or edge devices.

Considerations	Requires some familiarity with the model management model.
----------------	--

Azure Cognitive Services

[Azure Cognitive Services](#) is a set of *pre-built* APIs that enable you to build apps that use natural methods of communication. The term pre-built suggests that you do not need to bring datasets or data science expertise to train models to use in your applications. That's all done for you and packaged as APIs and SDKs that allow your apps to see, hear, speak, understand, and interpret user needs with just a few lines of code. You can easily add intelligent features to your apps, such as:

- **Vision:** Object detection, face recognition, OCR, etc. See [Computer Vision](#), [Face](#), [Form Recognizer](#).
- **Speech:** Speech-to-text, text-to-speech, speaker recognition, etc. See [Speech Service](#).
- **Language:** Translation, Sentiment analysis, key phrase extraction, language understanding, etc. See [Translator](#), [Text Analytics](#), [Language Understanding](#), [QnA Maker](#)
- **Decision:** Anomaly detection, content moderation, reinforcement learning. See [Anomaly Detector](#), [Content Moderator](#), [Personalizer](#).

Use Cognitive Services to develop apps across devices and platforms. The APIs keep improving, and are easy to set up.

Type	APIs for building intelligent applications
Supported languages	Various options depending on the service. Standard ones are C#, Java, JavaScript, and Python.
Machine learning phases	Deployment
Key benefits	Build intelligent applications using pre-trained models available through REST API and SDK. Variety of models for natural communication methods with vision, speech, language, and decision. No machine learning or data science expertise required.

SQL machine learning

[SQL machine learning](#) adds statistical analysis, data visualization, and predictive analytics in Python and R for relational data, both on-premises and in the cloud. Current platforms and tools include:

- [SQL Server Machine Learning Services](#)
- [Machine Learning Services on SQL Server Big Data Clusters](#)
- [Azure SQL Managed Instance Machine Learning Services](#)
- [Machine learning in Azure Synapse Analytics](#)
- [Machine learning and AI with ONNX in Azure SQL Edge](#)
- [Machine Learning extension for Azure Data Studio](#)

Use SQL machine learning when you need built-in AI and predictive analytics on relational data in SQL.

Type	On-premises predictive analytics for relational data
------	--

Supported languages	Python, R, SQL
Machine learning phases	Data preparation Model training Deployment
Key benefits	Encapsulate predictive logic in a database function, making it easy to include in data-tier logic.
Considerations	Assumes a SQL database as the data tier for your application.

Azure Data Science Virtual Machine

The [Azure Data Science Virtual Machine](#) is a customized virtual machine environment on the Microsoft Azure cloud. It is available in versions for both Windows and Linux Ubuntu. The environment is built specifically for doing data science and developing ML solutions. It has many popular data science, ML frameworks, and other tools pre-installed and pre-configured to jump-start building intelligent applications for advanced analytics.

Use the Data Science VM when you need to run or host your jobs on a single node. Or if you need to remotely scale up your processing on a single machine.

Type	Customized virtual machine environment for data science
Key benefits	<p>Reduced time to install, manage, and troubleshoot data science tools and frameworks.</p> <p>The latest versions of all commonly used tools and frameworks are included.</p> <p>Virtual machine options include highly scalable images with GPU capabilities for intensive data modeling.</p>
Considerations	<p>The virtual machine cannot be accessed when offline.</p> <p>Running a virtual machine incurs Azure charges, so you must be careful to have it running only when required.</p>

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform. Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts. Use Python, R, Scala, and SQL code in web-based notebooks to query, visualize, and model data.

Use Databricks when you want to collaborate on building machine learning solutions on Apache Spark.

Type	Apache Spark-based analytics platform
Supported languages	Python, R, Scala, SQL

Machine learning phases	Data preparation Data preprocessing Model training Model tuning Model inference Management Deployment
--------------------------------	---

ML.NET

[ML.NET](#) is an open-source, and cross-platform machine learning framework. With ML.NET, you can build custom machine learning solutions and integrate them into your .NET applications. ML.NET offers varying levels of interoperability with popular frameworks like TensorFlow and ONNX for training and scoring machine learning and deep learning models. For resource-intensive tasks like training image classification models, you can take advantage of Azure to train your models in the cloud.

Use ML.NET when you want to integrate machine learning solutions into your .NET applications. Choose between the [API](#) for a code-first experience and [Model Builder](#) or the [CLI](#) for a low-code experience.

Type	Open-source cross-platform framework for developing custom machine learning applications with .NET
Languages supported	C#, F#
Machine learning phases	Data preparation Training Deployment
Key benefits	Data science & ML experience not required Use familiar tools (Visual Studio, VS Code) and languages Deploy where .NET runs Extensible Scalable Local-first experience

Windows ML

[Windows ML](#) inference engine allows you to use trained machine learning models in your applications, evaluating trained models locally on Windows 10 devices.

Use Windows ML when you want to use trained machine learning models within your Windows applications.

Type	Inference engine for trained models in Windows devices
Languages supported	C#/C++, JavaScript

MMLSpark

[Microsoft ML for Apache Spark](#) (MMLSpark) is an open-source library that expands the distributed computing framework [Apache Spark](#). MMLSpark adds many deep learning and data science tools to the Spark ecosystem, including seamless integration of [Spark Machine Learning](#) pipelines with [Microsoft Cognitive Toolkit \(CNTK\)](#),

[LightGBM](#), [LIME \(Model Interpretability\)](#), and [OpenCV](#). You can use these tools to create powerful predictive models on any Spark cluster, such as [Azure Databricks](#) or [Cosmic Spark](#).

MMLSpark also brings new networking capabilities to the Spark ecosystem. With the HTTP on Spark project, users can embed any web service into their SparkML models. Additionally, MMLSpark provides easy-to-use tools for orchestrating [Azure Cognitive Services](#) at scale. For production-grade deployment, the Spark Serving project enables high throughput, submillisecond latency web services, backed by your Spark cluster.

Type	Open-source, distributed machine learning and microservices framework for Apache Spark
Languages supported	Scala 2.11, Java, Python 3.5+, R (beta)
Machine learning phases	Data preparation Model training Deployment
Key benefits	Scalability Streaming + Serving compatible Fault-tolerance
Considerations	Requires Apache Spark

Next steps

- To learn about all the Artificial Intelligence (AI) development products available from Microsoft, see [Microsoft AI platform](#).
- For training in developing AI and Machine Learning solutions with Microsoft, see [Microsoft Learn](#).

Machine learning at scale

3/10/2022 • 3 minutes to read • [Edit Online](#)

Machine learning (ML) is a technique used to train predictive models based on mathematical algorithms.

Machine learning analyzes the relationships between data fields to predict unknown values.

Creating and deploying a machine learning model is an iterative process:

- Data scientists explore the source data to determine relationships between *features* and predicted *labels*.
- The data scientists train and validate models based on appropriate algorithms to find the optimal model for prediction.
- The optimal model is deployed into production, as a web service or some other encapsulated function.
- As new data is collected, the model is periodically retrained to improve its effectiveness.

Machine learning at scale addresses two different scalability concerns. The first is training a model against large data sets that require the scale-out capabilities of a cluster to train. The second centers on operationalizing the learned model so it can scale to meet the demands of the applications that consume it. Typically this is accomplished by deploying the predictive capabilities as a web service that can then be scaled out.

Machine learning at scale has the benefit that it can produce powerful, predictive capabilities because better models typically result from more data. Once a model is trained, it can be deployed as a stateless, highly performant scale-out web service.

Model preparation and training

During the model preparation and training phase, data scientists explore the data interactively using languages like Python and R to:

- Extract samples from high volume data stores.
- Find and treat outliers, duplicates, and missing values to clean the data.
- Determine correlations and relationships in the data through statistical analysis and visualization.
- Generate new calculated features that improve the predictiveness of statistical relationships.
- Train ML models based on predictive algorithms.
- Validate trained models using data that was withheld during training.

To support this interactive analysis and modeling phase, the data platform must enable data scientists to explore data using a variety of tools. Additionally, the training of a complex machine learning model can require a lot of intensive processing of high volumes of data, so sufficient resources for scaling out the model training is essential.

Model deployment and consumption

When a model is ready to be deployed, it can be encapsulated as a web service and deployed in the cloud, to an edge device, or within an enterprise ML execution environment. This deployment process is referred to as operationalization.

Challenges

Machine learning at scale produces a few challenges:

- You typically need a lot of data to train a model, especially for deep learning models.

- You need to prepare these big data sets before you can even begin training your model.
- The model training phase must access the big data stores. It's common to perform the model training using the same big data cluster, such as Spark, that is used for data preparation.
- For scenarios such as deep learning, not only will you need a cluster that can provide you scale-out on CPUs, but your cluster will need to consist of GPU-enabled nodes.

Machine learning at scale in Azure

Before deciding which ML services to use in training and operationalization, consider whether you need to train a model at all, or if a prebuilt model can meet your requirements. In many cases, using a prebuilt model is just a matter of calling a web service or using an ML library to load an existing model. Some options include:

- Use the web services provided by Azure Cognitive Services.
- Use the pretrained neural network models provided by the Cognitive Toolkit.
- Embed the serialized models provided by Core ML for an iOS app.

If a prebuilt model does not fit your data or your scenario, options in Azure include Azure Machine Learning, HDInsight with Spark MLLib and MMLSpark, Azure Databricks, Cognitive Toolkit, and SQL Machine Learning Services. If you decide to use a custom model, you must design a pipeline that includes model training and operationalization.

For a list of technology choices for ML in Azure, see:

- [Choosing a cognitive services technology](#)
- [Choosing a machine learning technology](#)
- [Choosing a natural language processing technology](#)

Next steps

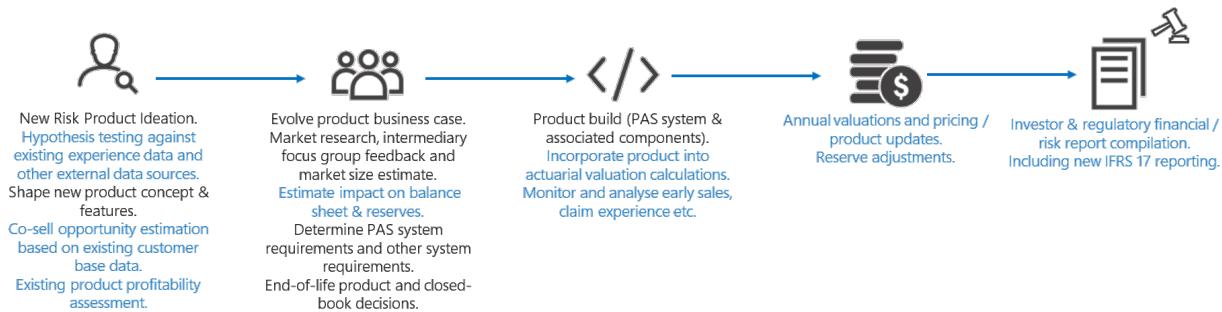
The following reference architectures show machine learning scenarios in Azure:

- [Batch scoring on Azure for deep learning models](#)
- [Real-time scoring of Python Scikit-Learn and Deep Learning Models on Azure](#)

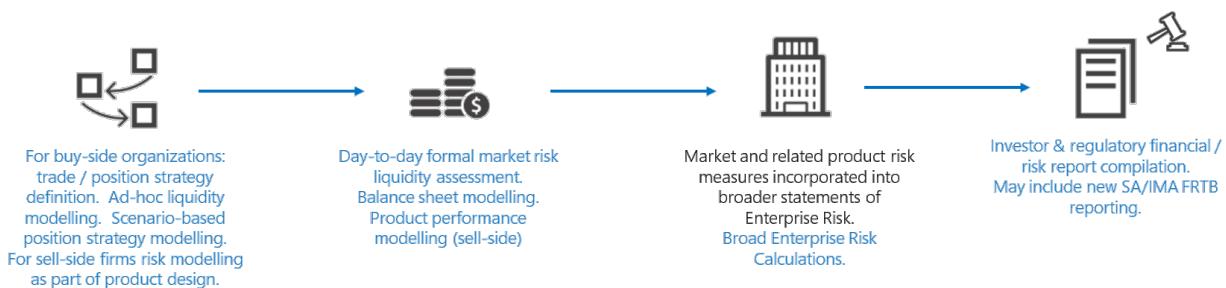
Enable the financial services risk lifecycle with Azure and R

3/10/2022 • 14 minutes to read • [Edit Online](#)

Risk calculations are pivotal at several stages in the lifecycle of key financial services operations. For example, a simplified form of the insurance product management lifecycle might look something like the diagram below. The risk calculation aspects are shown in blue text.



A scenario in a capital markets firm might look like this:



Through these processes, there are common needs around risk modeling including:

- The need for ad-hoc risk-related experimentation by risk analysts; actuaries in an insurance firm or quants in a capital markets firm. These analysts typically work with code and modeling tools popular in their domain: R and Python. Many university curriculums include training in R or Python in mathematical finance and in MBA courses. Both languages offer a wide range of open source libraries that support popular risk calculations.
- Along with appropriate tooling, analysts often require access to:
 - Accurate market pricing data.
 - Existing policy and claims data.
 - Existing market position data.
 - Other external data. Sources include structured data such as mortality tables and competitive pricing data. Less traditional sources such as weather, news and others may also be used.
 - Computational capacity to enable quick interactive data investigations.
- They may also make use of ad-hoc machine learning algorithms for pricing or determining market strategy.
- The need to visualize and present data for use in product planning, trading strategy, and similar discussions.

- The rapid execution of defined models, configured by the analysts for pricing, valuations, and market risk. The valuations use a combination of dedicated risk modeling, market risk tools, and custom code. The analysis is executed in a batch with varying nightly, weekly, monthly, quarterly, and annual calculations. This analysis generates spikes in workloads.
- The integration of data with other enterprise wide risk measures for consolidated risk reporting. In larger organizations, lower level risk estimates can be transferred to an enterprise risk modeling and reporting tool.
- Results must be reported in a defined format at the required interval to meet investor and regulatory requirements.

Microsoft supports the above concerns through a combination of Azure services and partner offerings in the [Azure Marketplace](#). In this article, we show practical examples of how to perform ad-hoc experimentation using R. We begin by explaining how to run the experiment on a single machine. Next, we show you how to run the same experiment on [Azure Batch](#), and we close by showing you how to take advantage of external services in our modeling. The options and considerations for the execution of defined models on Azure are described in these articles focused on [banking](#) and [insurance](#).

Analyst modeling in R

Let's start by looking at how R may be used by an analyst in a simplified, representative capital markets scenario. You can build this either by referencing an existing R library for the calculation or by writing code from scratch. In our example, we also must fetch external pricing data. To keep the example simple but illustrative, we calculate the potential future exposure (PFE) of an equity stock forward contract. This example avoids complex quantitative modeling techniques for instruments like complex derivatives and focuses on a single risk factor to concentrate on the risk life cycle. Our example lets you do the following actions:

- Select an instrument of interest.
- Source historic prices for the instrument.
- Model equity price by simple Monte Carlo (MC) calculation, which uses Geometric Brownian Motion (GBM):
 - Estimate expected return μ (mu) and volatility σ (theta).
 - Calibrate the model to historic data.
- Visualize the various paths to communicate the results.
- Plot $\max(0, \text{Stock Value})$ to demonstrate the meaning of PFE, the difference to Value at Risk (VaR).
 - To clarify: PFE = Share Price (T) -- Forward Contract Price K
- Take the 0.95 Quantile to get the PFE value at each time step / end of simulation period.

We'll calculate the potential future exposure for an equity forward based on Microsoft (MSFT) stock. As mentioned previously, to model the stock prices, historic prices for the MSFT stock are required so we can calibrate the model to historical data. There are many ways to acquire historical stock prices. In our example, we use a free version of a stock price service from an external service provider, [Quandl](#).

NOTE

The example uses the [WIKI Prices dataset](#) which can be used for learning concepts. For production usage of US based equities, Quandl recommends using the [End of Day US Stock Prices dataset](#).

To process the data and define the risk associated with the equity, we need to do the following things:

- Retrieve history data from the equity.
- Determine the expected return μ and volatility σ from the historic data.
- Model the underlying stock prices using some simulation.
- Run the model.
- Determine the exposure of the equity in the future.

We start by retrieving the stock from the Quandl service and plotting the closing price history over the last 180 days.

```
# Lubridate package must be installed
if (!require(lubridate)) install.packages('lubridate')
library(lubridate)

# Quandl package must be installed
if (!require(Quandl)) install.packages('Quandl')
library(Quandl)

# Get your API key from quandl.com
quandl_api = "enter your key here"

# Add the key to the Quandl keychain
Quandl.api_key(quandl_api)

quandl_get <-
  function(sym, start_date = "2018-01-01") {
    require(devtools)
    require(Quandl)
    # Retrieve the Open, High, Low, Close and Volume Column for a given Symbol
    # Column Indices can be deduced from this sample call
    # data <- Quandl(c("WIKI/MSFT"), rows = 1)

    tryCatch(Quandl(c(
      paste0("WIKI/", sym, ".8"),     # Column 8 : Open
      paste0("WIKI/", sym, ".9"),     # Column 9 : High
      paste0("WIKI/", sym, ".10"),    # Column 10: Low
      paste0("WIKI/", sym, ".11"),    # Column 11: Close
      paste0("WIKI/", sym, ".12")),   # Column 12: Volume
      start_date = start_date,
      type = "raw"
    ))
  }

# Define the Equity Forward
instrument.name <- "MSFT"
instrument.premium <- 100.1
instrument.pfeQuantile <- 0.95

# Get the stock price for the last 180 days
instrument.startDate <- today() - days(180)

# Get the quotes for an equity and transform them to a data frame
df_instrument.timeSeries <- quandl_get(instrument.name,start_date = instrument.startDate)

#Rename the columns
colnames(df_instrument.timeSeries) <- c()
colnames(df_instrument.timeSeries) <- c("Date","Open","High","Low","Close","Volume")

# Plot the closing price history to get a better feeling for the data
plot(df_instrument.timeSeries$Date, df_instrument.timeSeries$Close)
```

With the data in hand, we calibrate the GBM model.

```

# Code inspired by the book Computational Finance, An Introductory Course with R by
#     A. Arratia.

# Calculate the daily return in order to estimate sigma and mu in the Wiener Process
df_instrument.dailyReturns <- c(diff(log(df_instrument.timeSeries$Close)), NA)

# Estimate the mean of std deviation of the log returns to estimate the parameters of the Wiener Process

estimateGBM_Parameters <- function(logReturns, dt = 1/252) {

    # Volatility
    sigma_hat = sqrt(var(logReturns)) / sqrt(dt)

    # Drift
    mu_hat = mean(logReturns) / dt + sigma_hat**2 / 2.0

    # Return the parameters
    parameter.list <- list("mu" = mu_hat, "sigma" = sigma_hat)

    return(parameter.list)
}

# Calibrate the model to historic data
GBM_Parameters <- estimateGBM_Parameters(df_instrument.dailyReturns[1:length(df_instrument.dailyReturns) - 1])

```

Next, we model the underlying stock prices. We can either implement the discrete GBM process from scratch or utilize one of many R packages which provide this functionality. We use the R package [sde \(Simulation and Inference for Stochastic Differential Equations\)](#) which provides a method of solving this problem. The GBM method requires a set of parameters which are either calibrated to historic data or given as simulation parameters. We use the historic data, providing μ , σ and the stock prices at the beginning of the simulation (P_0).

```

if (!require(sde)) install.packages('sde')
library(sde)

sigma <- GBM_Parameters$sigma
mu <- GBM_Parameters$mu
P0 <- tail(df_instrument.timeSeries$Close, 1)

# Calculate the PFE looking one month into the future
T <- 1 / 12

# Consider nt MC paths
nt=50

# Divide the time interval T into n discrete time steps
n = 2 ^ 8

dt <- T / n
t <- seq(0,T,by=dt)

```

We're now ready to start a Monte Carlo simulation to model the potential exposure for some number of simulation paths. We'll limit the simulation to 50 Monte Carlo paths and 256 time steps. In preparation for scaling out the simulation and taking advantage of parallelization in R, the Monte Carlo simulation loop uses a `foreach` statement.

```

# Track the start time of the simulation
start_s <- Sys.time()

# Instead of a simple for loop to execute a simulation per MC path, call the
# simulation with the foreach package
# in order to demonstrate the similarity to the AzureBatch way to call the method.

library(foreach)
# Execute the MC simulation for the wiener process utilizing the GBM method from the sde package
exposure_mc <- foreach (i=1:nt, .combine = rbind ) %do% GBM(x = P0, r = mu, sigma = sigma, T = T, N = n)
rownames(exposure_mc) <- c()

# Track the end time of the simulation
end_s <- Sys.time()

# Duration of the simulation

difftime(end_s, start_s)

```

We've now simulated the price of the underlying MSFT stock. To calculate the exposure of the equity forward, we subtract the premium and limit the exposure to only positive values.

```

# Calculate the total Exposure as V_i(t) - K, put it to zero for negative exposures
pfe_mc <- pmax(exposure_mc - instrument.premium ,0)

ymax <- max(pfe_mc)
ymin <- min(pfe_mc)
plot(t, pfe_mc[1,], t = 'l', ylim = c(ymin, ymax), col = 1, ylab="Credit Exposure in USD", xlab="time t in
Years")
for (i in 2:nt) {
  lines(t, pfe_mc[i,], t = 'l', ylim = c(ymin, ymax), col = i)
}

```

The next two pictures show the result of the simulation. The first picture shows the Monte Carlo simulation of the underlying stock price for 50 paths. The second picture illustrates the underlying credit exposure for the equity forward after subtracting the premium of the equity forward and limiting the exposure to positive values.

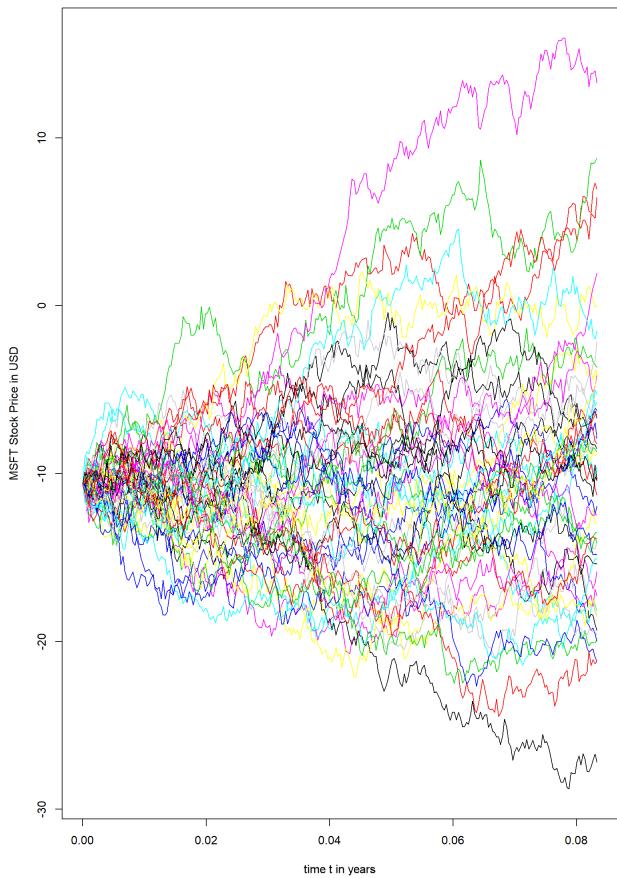


Figure 1 - 50 Monte Carlo paths

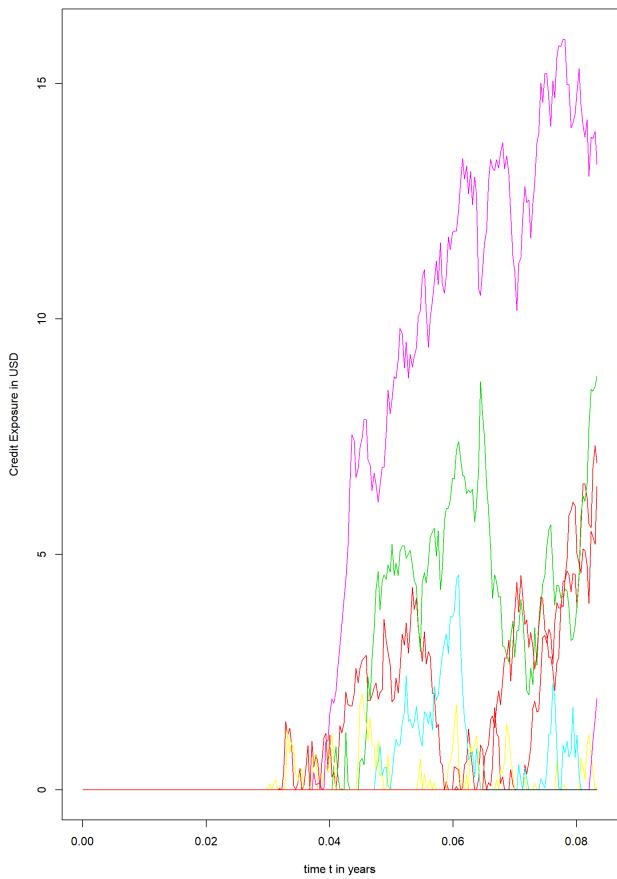


Figure 2 - Credit exposure for equity forward

In the last step, the 1-month 0.95 quantile PFE is calculated by the following code.

```

# Calculate the PFE at each time step
df_pfe <- cbind(t,apply(pfe_mc,2,quantile,probs = instrument.pfeQuantile ))

resulting in the final PFE plot
plot(df_pfe, t = 'l', ylab = "Potential Future Exposure in USD", xlab = "time t in Years")

```

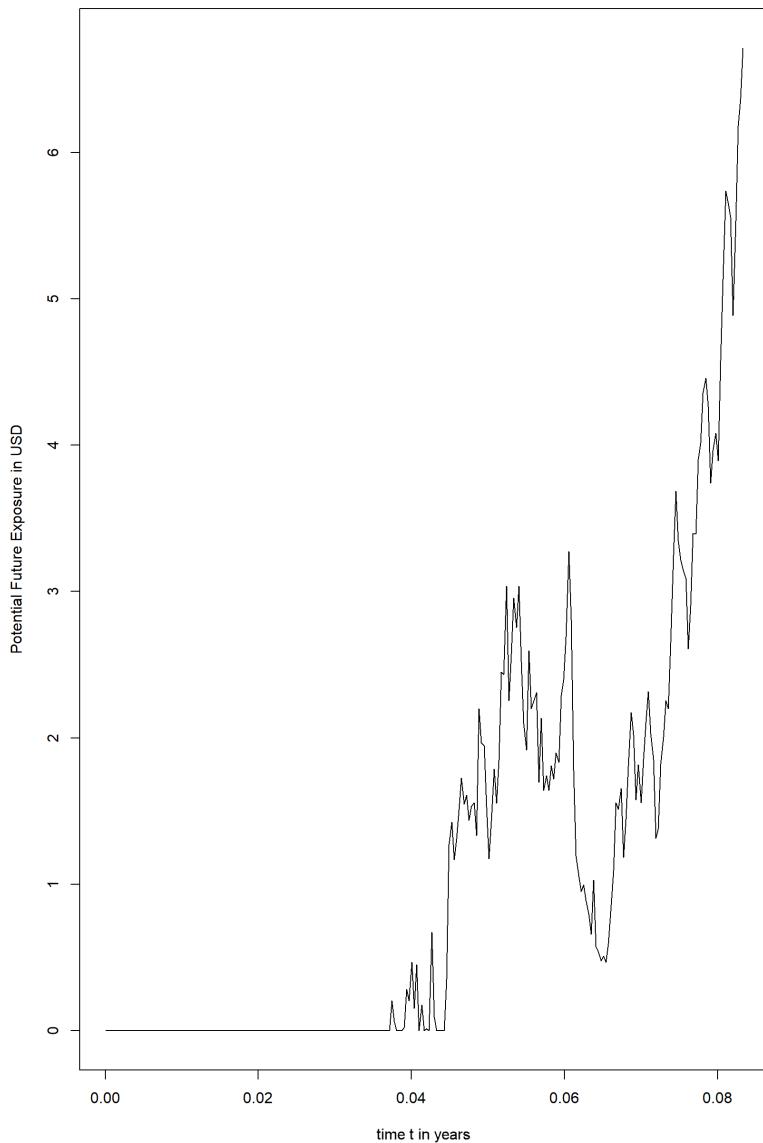


Figure 3 Potential future exposure for MSFT equity forward

Using Azure Batch with R

The R solution described above can be connected to Azure Batch and leverage the cloud for risk calculations. This takes little extra effort for a parallel calculation such as ours. The tutorial, [Run a parallel R simulation with Azure Batch](#), provides detailed information on connecting R to Azure Batch. Below we show the code and summary of the process to connect to Azure Batch and how to take advantage of the extension to the cloud in a simplified PFE calculation.

This example tackles the same model described earlier. As we've seen before, this calculation can run on our personal computer. Increases to the number of Monte Carlo paths or use of smaller time steps will result in much longer execution times. Almost all of the R code will remain unchanged. We'll highlight the differences in this section.

Each path of the Monte Carlo simulation runs in Azure. We can do this because each path is independent of the

others, giving us an "embarrassingly parallel" calculation.

To use Azure Batch, we define the underlying cluster and reference it in the code before the cluster can be used in the calculations. To run the calculations, we use the following cluster.json definition:

```
{  
  "name": "myMCPool",  
  "vmSize": "Standard_D2_v2",  
  "maxTasksPerNode": 4,  
  "poolSize": {  
    "dedicatedNodes": {  
      "min": 1,  
      "max": 1  
    },  
    "lowPriorityNodes": {  
      "min": 3,  
      "max": 3  
    },  
    "autoscaleFormula": "QUEUE"  
  },  
  "containerImage": "rocker/tidyverse:latest",  
  "rPackages": {  
    "cran": [],  
    "github": [],  
    "bioconductor": []  
  },  
  "commandLine": [],  
  "subnetId": ""  
}
```

With this cluster definition, the following R code makes use of the cluster:

```
# Define the cloud burst environment  
library(doAzureParallel)  
  
# set your credentials  
setCredentials("credentials.json")  
  
# Create your cluster if not exist  
cluster <- makeCluster("cluster.json")  
  
# register your parallel backend  
registerDoAzureParallel(cluster)  
  
# check that your workers are up  
getDoParWorkers()
```

Finally, we update the foreach statement from earlier to use the doAzureParallel package. It's a minor change, adding a reference to the sde package and changing the %do% to %dopar%:

```
# Execute the MC simulation for the wiener process utilizing the GBM method from the sde package and extend  
the computation to the cloud  
exposure_mc <- foreach(i = 1:nt, .combine = rbind, .packages = 'sde') %dopar% GBM(x = P0, r = mu, sigma =  
sigma, T = T, N = n)  
rownames(exposure_mc) <- c()
```

Each Monte Carlo simulation is submitted as a task to Azure Batch. The task executes in the cloud. Results are merged before being sent back to the analyst workbench. The heavy lifting and computations execute in the cloud to take full advantage of scaling and the underlying infrastructure required by the requested calculations.

After the calculations have finished, the additional resources can easily be shut-down by invoking the following

a single instruction:

```
# Stop the cloud cluster  
stopCluster(cluster)
```

Use a SaaS offering

The first two examples show how to utilize local and cloud infrastructure for developing an adequate valuation model. This paradigm has begun to shift. In the same way that on-premises infrastructure has transformed into cloud-based IaaS and PaaS services, the modeling of relevant risk figures is transforming into a service-oriented process. Today's analysts face two major challenges:

- The regulatory requirements use increasing compute capacity to add to modeling requirements. The regulators are asking for more frequent and up-to date risk figures.
- The existing risk infrastructure has grown organically with time and creates challenges when implementing new requirements and more advanced risk modeling in an agile manner.

Cloud-based services can deliver the required functionality and support risk analysis. This approach has some advantages:

- The most common risk calculations required by the regulator must be implemented by everyone under the regulation. By utilizing services from a specialized service provider, the analyst benefits from ready to use, regulator-compliant risk calculations. Such services may include market risk calculations, counterparty risk calculations, X-Value Adjustment (XVA), and even Fundamental Review of Trading Book (FRTB) calculations.
- These services expose their interfaces through web services. The existing risk infrastructure can be enhanced by these other services.

In our example, we want to invoke a cloud-based service for FRTB calculations. Several of these can be found on [AppSource](#). For this article we chose a trial option from [Vector Risk](#). We'll continue to modify our system. This time, we use a service to calculate the risk figure of interest. This process consists of the following steps:

1. Call the relevant risk service and with the right parameters.
2. Wait until the service finishes the calculation.
3. Retrieve and incorporate the results into the risk analysis.

Translated into R code, our R code can be enhanced by the definition of the required input values from a prepared input template.

```
Template <- readLines('RequiredInputData.json')  
data <- list()  
# drilldown setup  
timeSteps = seq(0, n, by = 1),  
paths = as.integer(seq(0, nt, length.out = min(nt, 100))),  
# calc setup  
calcDate = instrument.startDate,  
npaths = nt,  
price = P0,  
vol = sigma,  
drift = mu,  
premium = instrument.premium,  
maturityDate = today()  
)  
body <- whisker.render(template, data)
```

Next, we need to call the web service. In this case, we call the StartCreditExposure method to trigger the calculation. We store the endpoint for the API in a variable named *endpoint*.

```
# make the call
result <- POST( paste(endpoint, "StartCreditExposure", sep = ""),
                 authenticate(username, password, type = "basic"),
                 content_type("application/json"),
                 add_headers(`Ocp-Apim-Subscription-Key` = api_key),
                 body = body, encode = "raw"
               )

result <- content(result)
```

Once the calculations have finished, we retrieve the results.

```
# get back high level results
result <- POST( paste(endpoint, "GetCreditExposureResults", sep = ""),
                 authenticate(username, password, type = "basic"),
                 content_type("application/json"),
                 add_headers(`Ocp-Apim-Subscription-Key` = api_key),
                 body = sprintf('{"getCreditExposureResults":',
                               {"token": "DataSource=Production;Organisation=Microsoft", "ticket": "%s"}), ticket), encode = "raw")

result <- content(result)
```

This leaves the analyst to continue with the results received. The relevant risk figures of interest are extracted from the results and plotted.

```
if (!is.null(result$error)) {
  cat(result$error$message)
} else {
  # plot PFE
  result <- result$getCreditExposureResultsResponse$getCreditExposureResultsResult
  df <- do.call(rbind, result$exposures)
  df <- as.data.frame(df)
  df <- subset(df, term <= n)
}

plot(as.numeric(df$term[df$statistic == 'PFE']) / 365, df$result[df$statistic == 'PFE'], type = "p", xlab =
  ("time t in Years"), ylab = ("Potential Future Exposure in USD"), ylim = range(c(df$result[df$statistic ==
  'PFE'], df$result[df$statistic == 'PFE'])), col = "red")
```

The resulting plots look like this:

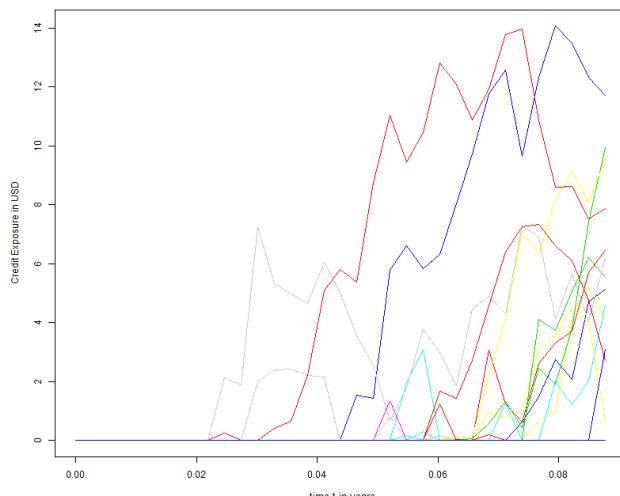


Figure 4 - Credit exposure for MSFT equity forward - Calculated with a cloud-based risk engine

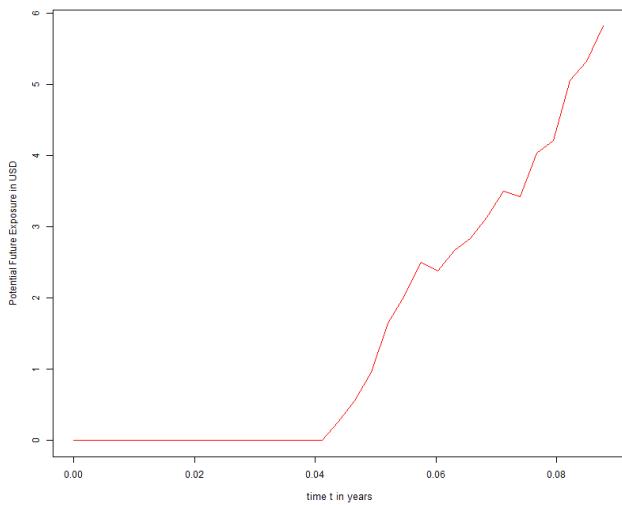


Figure 5 - Potential future exposure for MSFT equity forward - Calculated with a cloud-based risk engine

Contributors

This article is being updated and maintained by Microsoft. It was originally written by the following contributors:

- [Dr. Darko Mocelj](#) | HPC Global Blackbelt & AI Sr. Technology Specialist
- [Rupert Nicolay](#) | Financial Services Industry Solutions Lead

Next steps

Flexible access to the cloud through compute infrastructure and SaaS-based risk analysis services can deliver improvements in speed and agility for risk analysts working in capital markets and insurance. In this article we worked through an example which illustrates how to use Azure and other services using tools risk analysts know. Try taking advantage of Azure's capabilities as you create and enhance your risk models.

Tutorials

- R developers: [Run a parallel R simulation with Azure Batch](#)
- [Basic R commands and RevoScaleR functions: 25 common examples](#)
- [Visualize and analyze data using RevoScaleR](#)
- [Introduction to ML services and open-source R capabilities on HDInsight](#)

The Team Data Science Process lifecycle

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process (TDSP) provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the complete steps that successful projects follow. If you use another data-science lifecycle, such as the Cross Industry Standard Process for Data Mining ([CRISP-DM](#)), Knowledge Discovery in Databases ([KDD](#)), or your organization's own custom process, you can still use the task-based TDSP.

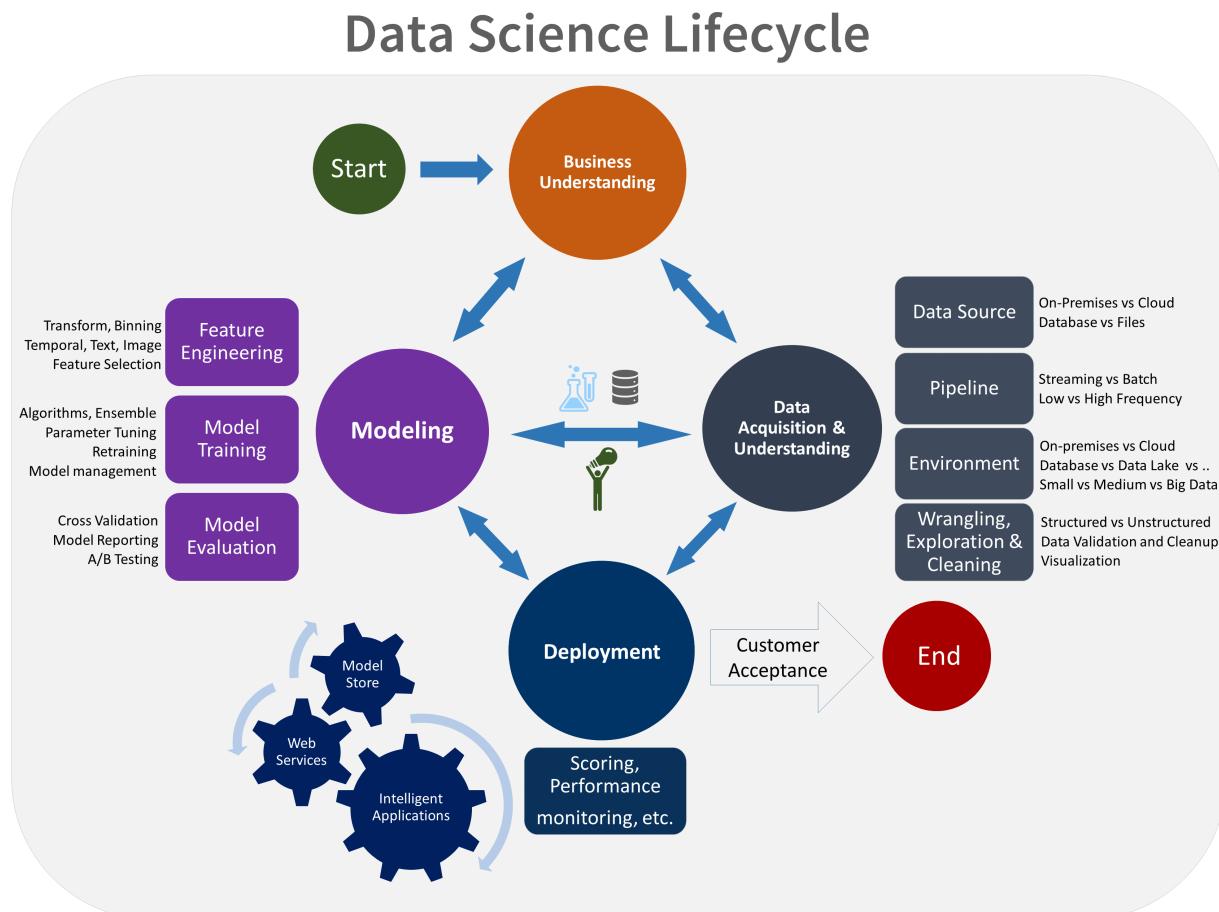
This lifecycle is designed for data-science projects that are intended to ship as part of intelligent applications. These applications deploy machine learning or artificial intelligence models for predictive analytics. Exploratory data-science projects and improvised analytics projects can also benefit from the use of this process. But for those projects, some of the steps described here might not be needed.

Five lifecycle stages

The TDSP lifecycle is composed of five major stages that are executed iteratively. These stages include:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

Here is a visual representation of the TDSP lifecycle:



The TDSP lifecycle is modeled as a sequence of iterated steps that provide guidance on the tasks needed to use

predictive models. You deploy the predictive models in the production environment that you plan to use to build the intelligent applications. The goal of this process lifecycle is to continue to move a data-science project toward a clear engagement end point. Data science is an exercise in research and discovery. The ability to communicate tasks to your team and your customers by using a well-defined set of artifacts that employ standardized templates helps to avoid misunderstandings. Using these templates also increases the chance of the successful completion of a complex data-science project.

For each stage, we provide the following information:

- **Goals:** The specific objectives.
- **How to do it:** An outline of the specific tasks and guidance on how to complete them.
- **Artifacts:** The deliverables and the support to produce them.

Next steps

For examples of how to execute steps in TDSPs that use Azure Machine Learning, see [Use the TDSP with Azure Machine Learning](#).

Related resources

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

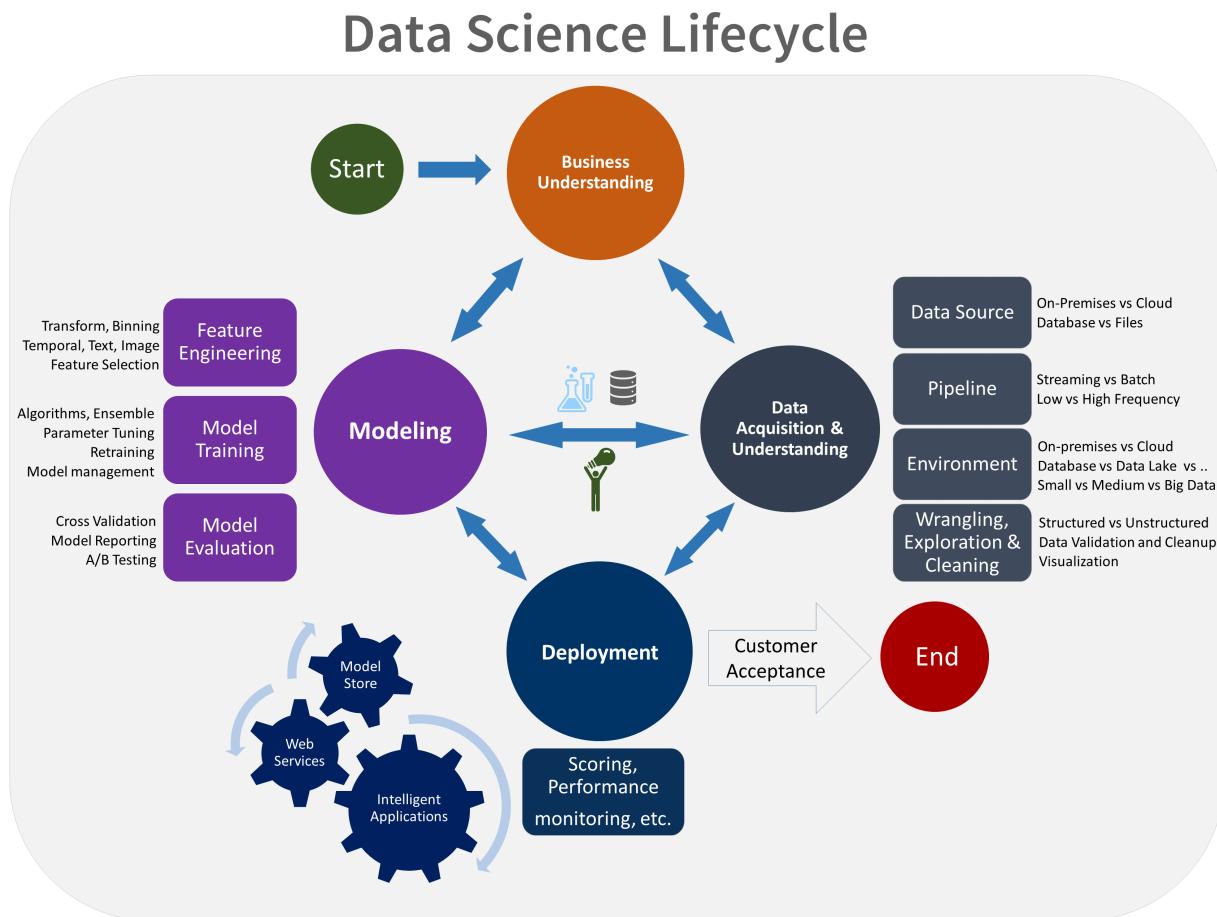
The business understanding stage of the Team Data Science Process lifecycle

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the business understanding stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

Here is a visual representation of the TDSP lifecycle:



Goals

- Specify the key variables that are to serve as the model targets and whose related metrics are used determine the success of the project.
- Identify the relevant data sources that the business has access to or needs to obtain.

How to do it

There are two main tasks addressed in this stage:

- **Define objectives:** Work with your customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.
- **Identify data sources:** Find the relevant data that helps you answer the questions that define the objectives of the project.

Define objectives

1. A central objective of this step is to identify the key business variables that the analysis needs to predict. We refer to these variables as the *model targets*, and we use the metrics associated with them to determine the success of the project. Two examples of such targets are sales forecasts or the probability of an order being fraudulent.

2. Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. You typically use data science or machine learning to answer five types of questions:

- How much or how many? (regression)
- Which category? (classification)
- Which group? (clustering)
- Is this weird? (anomaly detection)
- Which option should be taken? (recommendation)

Determine which of these questions you're asking and how answering it achieves your business goals.

3. Define the project team by specifying the roles and responsibilities of its members. Develop a high-level milestone plan that you iterate on as you discover more information.

4. Define the success metrics. For example, you might want to achieve a customer churn prediction. You need an accuracy rate of "x" percent by the end of this three-month project. With this data, you can offer customer promotions to reduce churn. The metrics must be **SMART**:

- Specific
- Measurable
- Achievable
- Relevant
- Time-bound

Identify data sources

Identify data sources that contain known examples of answers to your sharp questions. Look for the following data:

- Data that's relevant to the question. Do you have measures of the target and features that are related to the target?
- Data that's an accurate measure of your model target and the features of interest.

For example, you might find that the existing systems need to collect and log additional kinds of data to address the problem and achieve the project goals. In this situation, you might want to look for external data sources or update your systems to collect new data.

Artifacts

Here are the deliverables in this stage:

- **Charter document:** A standard template is provided in the TDSP project structure definition. The charter

document is a living document. You update the template throughout the project as you make new discoveries and as business requirements change. The key is to iterate upon this document, adding more detail, as you progress through the discovery process. Keep the customer and other stakeholders involved in making the changes and clearly communicate the reasons for the changes to them.

- **Data sources:** The **Raw data sources** section of the **Data definitions** report that's found in the TDSP project **Data report** folder contains the data sources. This section specifies the original and destination locations for the raw data. In later stages, you fill in additional details like the scripts to move the data to your analytic environment.
- **Data dictionaries:** This document provides descriptions of the data that's provided by the client. These descriptions include information about the schema (the data types and information on the validation rules, if any) and the entity-relation diagrams, if available.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

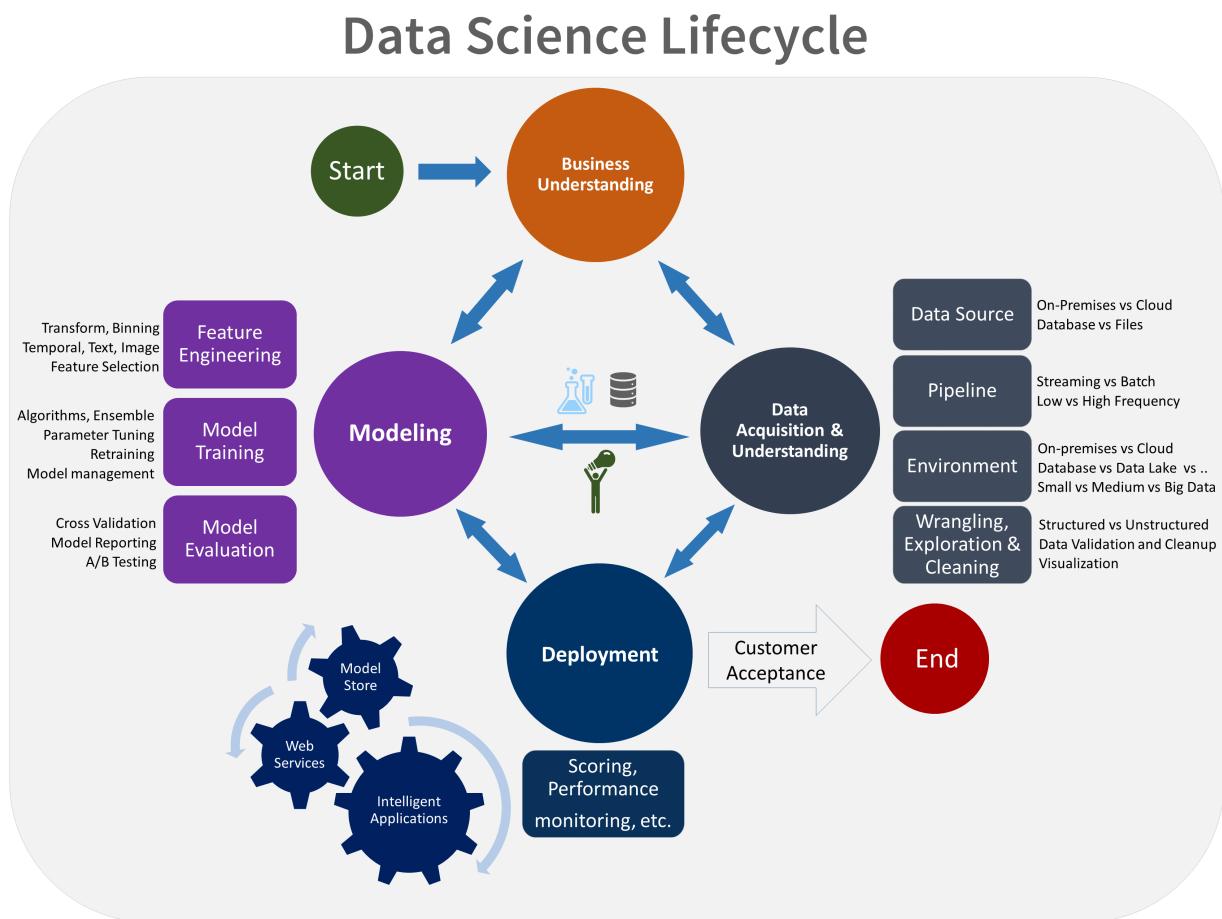
Data acquisition and understanding stage of the Team Data Science Process

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the data acquisition and understanding stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

Here is a visual representation of the TDSP lifecycle:



Goals

- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so you are ready to model.
- Develop a solution architecture of the data pipeline that refreshes and scores the data regularly.

How to do it

There are three main tasks addressed in this stage:

- **Ingest the data** into the target analytic environment.
- **Explore the data** to determine if the data quality is adequate to answer the question.
- **Set up a data pipeline** to score new or regularly refreshed data.

Ingest the data

Set up the process to move the data from the source locations to the target locations where you run analytics operations, like training and predictions. For technical details and options on how to move the data with various Azure data services, see [Load data into storage environments for analytics](#).

Explore the data

Before you train your models, you need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. You can use data summarization and visualization to audit the quality of your data and provide the information you need to process the data before it's ready for modeling. This process is often iterative. For guidance on cleaning the data, see [Tasks to prepare data for enhanced machine learning](#).

After you're satisfied with the quality of the cleansed data, the next step is to better understand the patterns that are inherent in the data. This data analysis helps you choose and develop an appropriate predictive model for your target. Look for evidence for how well connected the data is to the target. Then determine whether there is sufficient data to move forward with the next modeling steps. Again, this process is often iterative. You might need to find new data sources with more accurate or more relevant data to augment the data set initially identified in the previous stage.

Set up a data pipeline

In addition to the initial ingestion and cleaning of the data, you typically need to set up a process to score new data or refresh the data regularly as part of an ongoing learning process. Scoring may be completed with a data pipeline or workflow. The [Move data from a SQL Server instance to Azure SQL Database with Azure Data Factory](#) article gives an example of how to set up a pipeline with [Azure Data Factory](#).

In this stage, you develop a solution architecture of the data pipeline. You develop the pipeline in parallel with the next stage of the data science project. Depending on your business needs and the constraints of your existing systems into which this solution is being integrated, the pipeline can be one of the following options:

- Batch-based
- Streaming or real time
- A hybrid

Artifacts

The following are the deliverables in this stage:

- **Data quality report:** This report includes data summaries, the relationships between each attribute and target, variable ranking, and more.
- **Solution architecture:** The solution architecture can be a diagram or description of your data pipeline that you use to run scoring or predictions on new data after you have built a model. It also contains the pipeline to retrain your model based on new data. Store the document in the [Project](#) directory when you use the TDSP directory structure template.
- **Checkpoint decision:** Before you begin full-feature engineering and model building, you can reevaluate the project to determine whether the value expected is sufficient to continue pursuing it. You might, for example, be ready to proceed, need to collect more data, or abandon the project as the data does not exist to answer the question.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

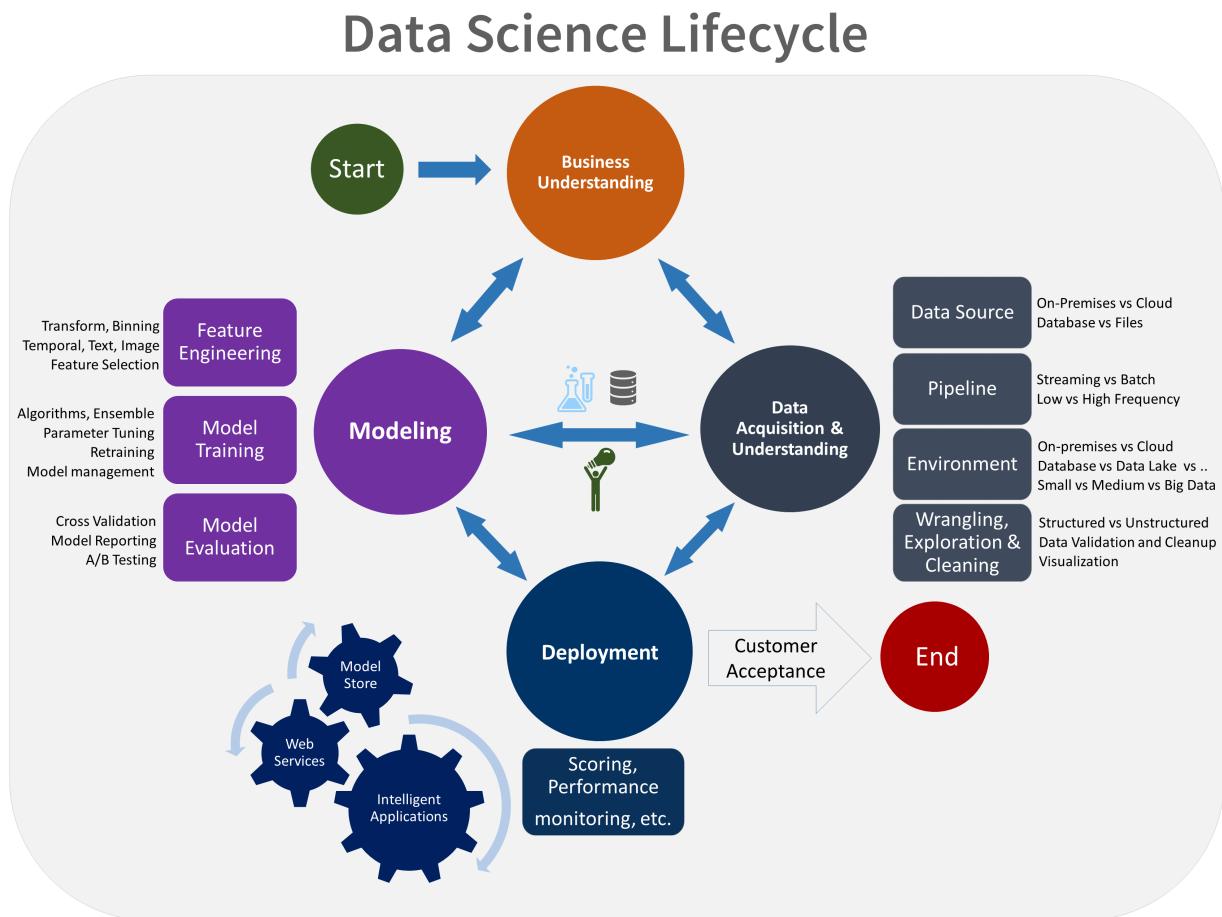
Modeling stage of the Team Data Science Process lifecycle

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the modeling stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

Here is a visual representation of the TDSP lifecycle:



Goals

- Determine the optimal data features for the machine-learning model.
- Create an informative machine-learning model that predicts the target most accurately.
- Create a machine-learning model that's suitable for production.

How to do it

There are three main tasks addressed in this stage:

- **Feature engineering:** Create data features from the raw data to facilitate model training.
- **Model training:** Find the model that answers the question most accurately by comparing their success metrics.
- Determine if your model is **suitable for production**.

Feature engineering

Feature engineering involves the inclusion, aggregation, and transformation of raw variables to create the features used in the analysis. If you want insight into what is driving a model, then you need to understand how the features relate to each other and how the machine-learning algorithms are to use those features.

This step requires a creative combination of domain expertise and the insights obtained from the data exploration step. Feature engineering is a balancing act of finding and including informative variables, but at the same time trying to avoid too many unrelated variables. Informative variables improve your result; unrelated variables introduce unnecessary noise into the model. You also need to generate these features for any new data obtained during scoring. As a result, the generation of these features can only depend on data that's available at the time of scoring.

Model training

Depending on the type of question that you're trying to answer, there are many modeling algorithms available. For guidance on choosing a prebuilt algorithm with designer, see [Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer](#); other algorithms are available through open-source packages in R or Python. Although this article focuses on Azure Machine Learning, the guidance it provides is useful for any machine-learning projects.

The process for model training includes the following steps:

- **Split the input data** randomly for modeling into a training data set and a test data set.
- **Build the models** by using the training data set.
- **Evaluate** the training and the test data set. Use a series of competing machine-learning algorithms along with the various associated tuning parameters (known as a *parameter sweep*) that are geared toward answering the question of interest with the current data.
- **Determine the "best" solution** to answer the question by comparing the success metrics between alternative methods.

See [Train models with Azure Machine Learning](#) for options on training models in Azure Machine Learning.

NOTE

Avoid leakage: You can cause data leakage if you include data from outside the training data set that allows a model or machine-learning algorithm to make unrealistically good predictions. Leakage is a common reason why data scientists get nervous when they get predictive results that seem too good to be true. These dependencies can be hard to detect. To avoid leakage often requires iterating between building an analysis data set, creating a model, and evaluating the accuracy of the results.

Model Evaluation

After training, the data scientist focuses next on model evaluation.

- **Checkpoint decision:** Evaluate whether the model performs sufficiently for production. Some key questions to ask are:
 - Does the model answer the question with sufficient confidence given the test data?
 - Should you try any alternative approaches?

- Should you collect additional data, do more feature engineering, or experiment with other algorithms?
- **Interpreting the Model:** Use [the Azure Machine Learning Python SDK](#) to perform the following tasks:
 - Explain the entire model behavior or individual predictions on your personal machine locally.
 - Enable interpretability techniques for engineered features.
 - Explain the behavior for the entire model and individual predictions in Azure.
 - Upload explanations to Azure Machine Learning Run History.
 - Use a visualization dashboard to interact with your model explanations, both in a Jupyter notebook and in the Azure Machine Learning workspace.
 - Deploy a scoring explainer alongside your model to observe explanations during inferencing.
- **Assessing Fairness:** The [Fairlearn open-source Python package with Azure Machine Learning](#) performs the following tasks:
 - Assess the fairness of your model predictions. This process will help you learn more about fairness in machine learning.
 - Upload, list, and download fairness assessment insights to/from Azure Machine Learning studio.
 - See the fairness assessment dashboard in Azure Machine Learning studio to interact with your model(s)' fairness insights.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

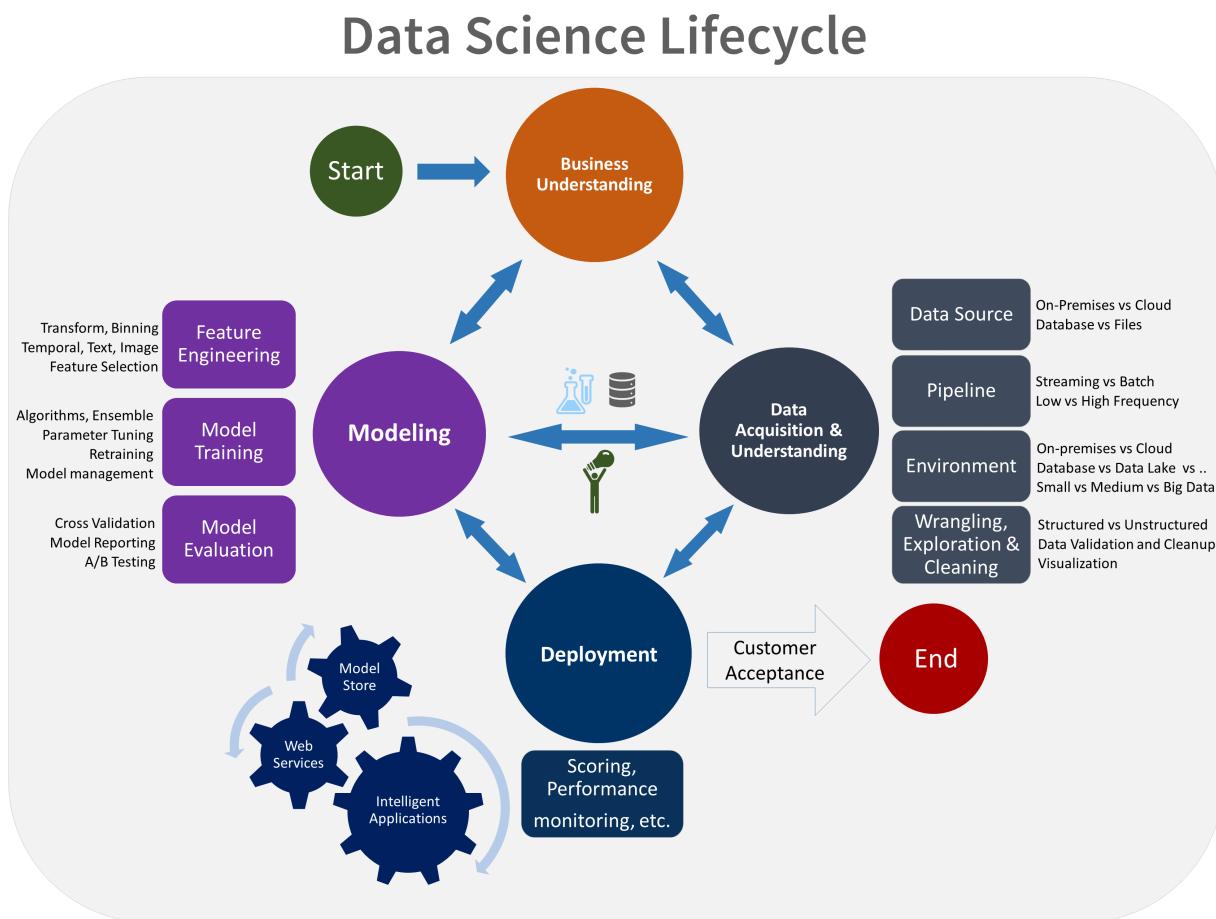
Deployment stage of the Team Data Science Process lifecycle

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the deployment of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

Here is a visual representation of the TDSP lifecycle:



Goal

Deploy models with a data pipeline to a production or production-like environment for final user acceptance.

How to do it

The main task addressed in this stage:

Operationalize the model: Deploy the model and pipeline to a production or production-like environment for

application consumption.

Operationalize a model

After you have a set of models that perform well, you can operationalize them for other applications to consume. Depending on the business requirements, predictions are made either in real time or on a batch basis. To deploy models, you expose them with an open API interface. The interface enables the model to be easily consumed from various applications, such as:

- Online websites
- Spreadsheets
- Dashboards
- Line-of-business applications
- Back-end applications

For examples of model operationalization with Azure Machine Learning, see [Deploy machine learning models to Azure](#). It is a best practice to build telemetry and monitoring into the production model and the data pipeline that you deploy. This practice helps with subsequent system status reporting and troubleshooting.

Artifacts

- A status dashboard that displays the system health and key metrics
- A final modeling report with deployment details
- A final solution architecture document

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data Acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

For Azure, we recommend applying TDSP using Azure Machine Learning: for an overview of Azure Machine Learning see [What is Azure Machine Learning?](#).

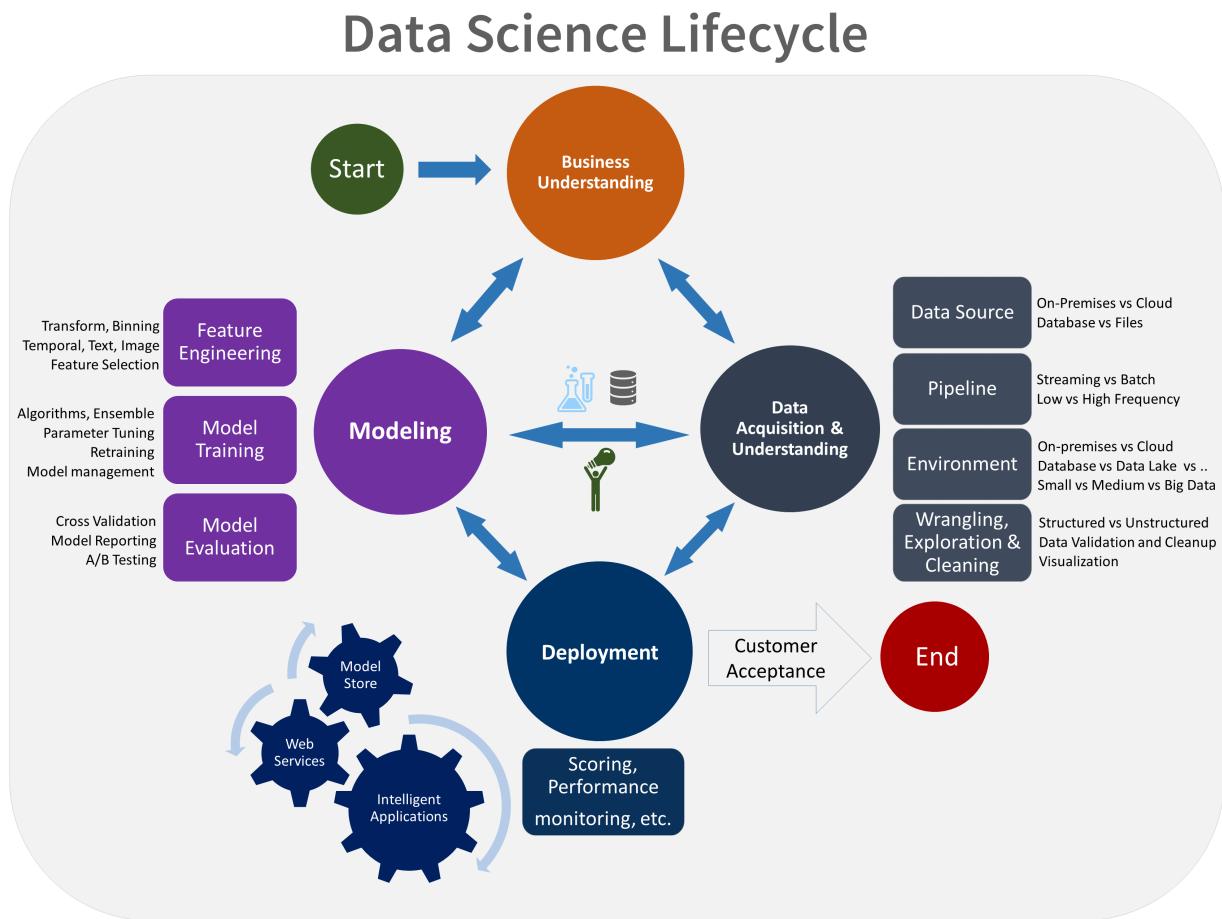
Customer acceptance stage of the Team Data Science Process lifecycle

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the customer acceptance stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

Here is a visual representation of the TDSP lifecycle:



Goal

Finalize project deliverables: Confirm that the pipeline, the model, and their deployment in a production environment satisfy the customer's objectives.

How to do it

There are two main tasks addressed in this stage:

- **System validation:** Confirm that the deployed model and pipeline meet the customer's needs.
- **Project hand-off:** Hand the project off to the entity that's going to run the system in production.

The customer should validate that the system meets their business needs and that it answers the questions with acceptable accuracy to deploy the system to production for use by their client's application. All the documentation is finalized and reviewed. The project is handed-off to the entity responsible for operations. This entity might be, for example, an IT or customer data-science team or an agent of the customer that's responsible for running the system in production.

Artifacts

The main artifact produced in this final stage is the **Exit report of the project for the customer**. This technical report contains all the details of the project that are useful for learning about how to operate the system. TDSP provides an [Exit report template](#). You can use the template as is, or you can customize it for specific client needs.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

For Azure, we recommend applying TDSP using Azure Machine Learning: for an overview of Azure Machine Learning see [What is Azure Machine Learning?](#).

Team Data Science Process roles and tasks

3/10/2022 • 6 minutes to read • [Edit Online](#)

The Team Data Science Process (TDSP) is a framework developed by Microsoft that provides a structured methodology to efficiently build predictive analytics solutions and intelligent applications. This article outlines the key personnel roles and associated tasks for a data science team standardizing on this process.

This introductory article links to tutorials on how to set up the TDSP environment. The tutorials provide detailed guidance for using Azure DevOps Projects, Azure Repos repositories, and Azure Boards. The motivating goal is moving from concept through modeling and into deployment.

The tutorials use Azure DevOps because that is how to implement TDSP at Microsoft. Azure DevOps facilitates collaboration by integrating role-based security, work item management and tracking, and code hosting, sharing, and source control. The tutorials also use an Azure [Data Science Virtual Machine](#) (DSVM) as the analytics desktop, which has several popular data science tools pre-configured and integrated with Microsoft software and Azure services.

You can use the tutorials to implement TDSP using other code-hosting, agile planning, and development tools and environments, but some features may not be available.

Structure of data science groups and teams

Data science functions in enterprises are often organized in the following hierarchy:

- Data science group
 - Data science team/s within the group

In such a structure, there are group leads and team leads. Typically, a data science project is done by a data science team. Data science teams have project leads for project management and governance tasks, and individual data scientists and engineers to perform the data science and data engineering parts of the project. The initial project setup and governance is done by the group, team, or project leads.

Definition and tasks for the four TDSP roles

With the assumption that the data science unit consists of teams within a group, there are four distinct roles for TDSP personnel:

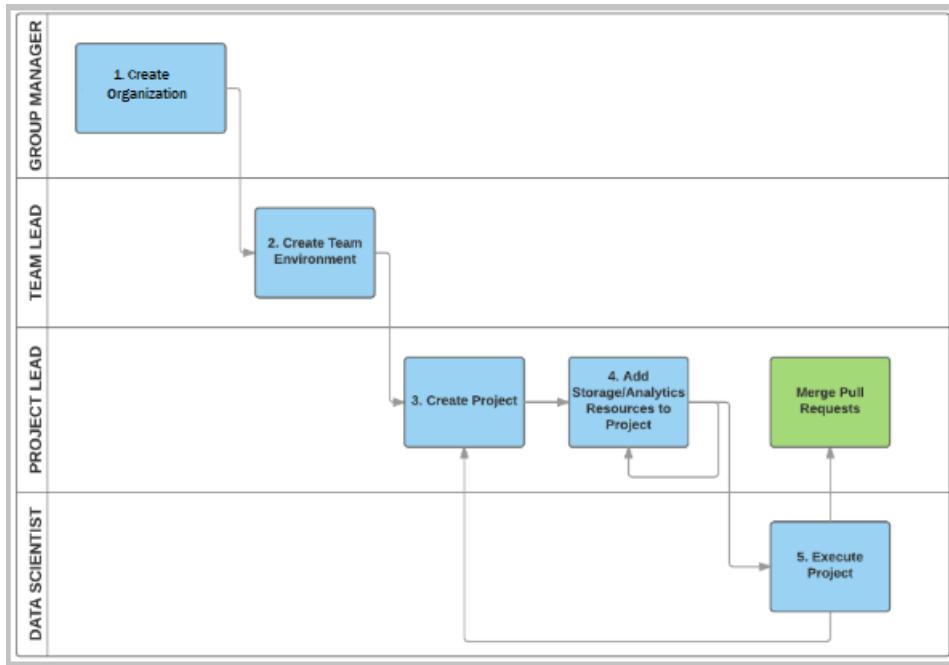
1. **Group Manager:** Manages the entire data science unit in an enterprise. A data science unit might have multiple teams, each of which is working on multiple data science projects in distinct business verticals. A Group Manager might delegate their tasks to a surrogate, but the tasks associated with the role do not change.
2. **Team Lead:** Manages a team in the data science unit of an enterprise. A team consists of multiple data scientists. For a small data science unit, the Group Manager and the Team Lead might be the same person.
3. **Project Lead:** Manages the daily activities of individual data scientists on a specific data science project.
4. **Project Individual Contributors:** Data Scientists, Business Analysts, Data Engineers, Architects, and others who execute a data science project.

NOTE

Depending on the structure and size of an enterprise, a single person may play more than one role, or more than one person may fill a role.

Tasks to be completed by the four roles

The following diagram shows the top-level tasks for each Team Data Science Process role. This schema and the following, more detailed outline of tasks for each TDSP role can help you choose the tutorial you need based on your responsibilities.



Group Manager tasks

The Group Manager or a designated TDSP system administrator completes the following tasks to adopt the TDSP:

- Creates an Azure DevOps **organization** and a group project within the organization.
- Creates a **project template repository** in the Azure DevOps group project, and seeds it from the project template repository developed by the Microsoft TDSP team. The Microsoft TDSP project template repository provides:
 - A **standardized directory structure**, including directories for data, code, and documents.
 - A set of **standardized document templates** to guide an efficient data science process.
- Creates a **utility repository**, and seeds it from the utility repository developed by the Microsoft TDSP team. The TDSP utility repository from Microsoft provides a set of useful utilities to make the work of a data scientist more efficient. The Microsoft utility repository includes utilities for interactive data exploration, analysis, reporting, and baseline modeling and reporting.
- Sets up the **security control policy** for the organization account.

For detailed instructions, see [Group Manager tasks for a data science team](#).

Team Lead tasks

The Team Lead or a designated project administrator completes the following tasks to adopt the TDSP:

- Creates a team **project** in the group's Azure DevOps organization.
- Creates the **project template repository** in the project, and seeds it from the group project template

repository set up by the Group Manager or delegate.

- Creates the **team utility repository**, seeds it from the group utility repository, and adds team-specific utilities to the repository.
- Optionally creates **Azure file storage** to store useful data assets for the team. Other team members can mount this shared cloud file store on their analytics desktops.
- Optionally mounts the Azure file storage on the team's **DSVM** and adds team data assets to it.
- Sets up **security control** by adding team members and configuring their permissions.

For detailed instructions, see [Team Lead tasks for a data science team](#).

Project Lead tasks

The Project Lead completes the following tasks to adopt the TDSP:

- Creates a **project repository** in the team project, and seeds it from the project template repository.
- Optionally creates **Azure file storage** to store the project's data assets.
- Optionally mounts the Azure file storage to the **DSVM** and adds project data assets to it.
- Sets up **security control** by adding project members and configuring their permissions.

For detailed instructions, see [Project Lead tasks for a data science team](#).

Project Individual Contributor tasks

The Project Individual Contributor, usually a Data Scientist, conducts the following tasks using the TDSP:

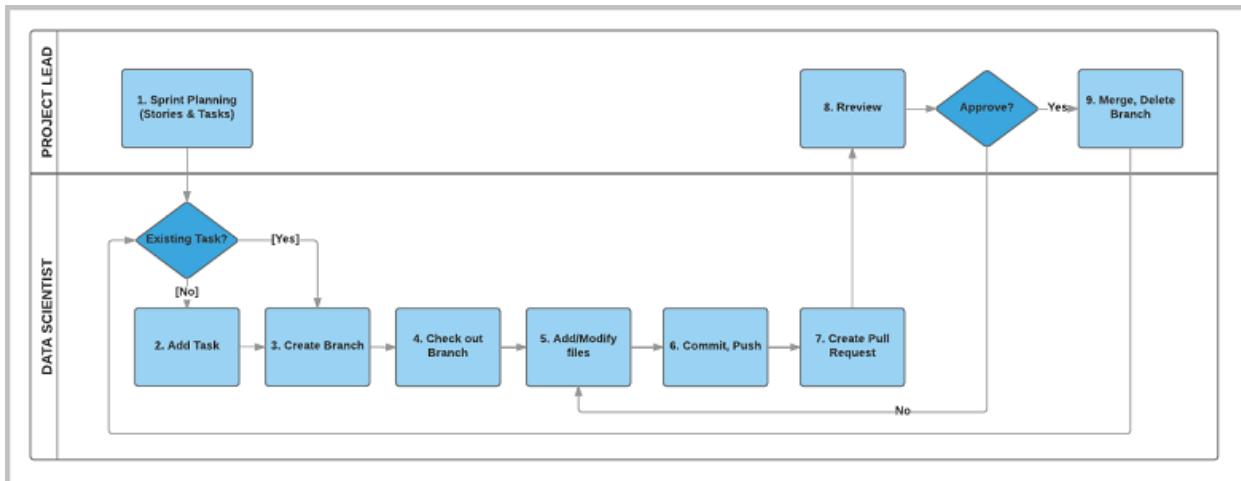
- Clones the **project repository** set up by the project lead.
- Optionally mounts the shared team and project **Azure file storage** on their **Data Science Virtual Machine** (DSVM).
- Executes the project.

For detailed instructions for onboarding onto a project, see [Project Individual Contributor tasks for a data science team](#).

Data science project execution workflow

By following the relevant tutorials, data scientists, project leads, and team leads can create work items to track all tasks and stages for project from beginning to end. Using Azure Repos promotes collaboration among data scientists and ensures that the artifacts generated during project execution are version controlled and shared by all project members. Azure DevOps lets you link your Azure Boards work items with your Azure Repos repository branches and easily track what has been done for a work item.

The following figure outlines the TDSP workflow for project execution:



The workflow steps can be grouped into three activities:

- Project Leads conduct sprint planning
- Data Scientists develop artifacts on `git` branches to address work items
- Project Leads or other team members do code reviews and merge working branches to the primary branch

For detailed instructions on project execution workflow, see [Agile development of data science projects](#).

TDSP project template repository

Use the Microsoft TDSP team's [project template repository](#) to support efficient project execution and collaboration. The repository gives you a standardized directory structure and document templates you can use for your own TDSP projects.

Next steps

Explore more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

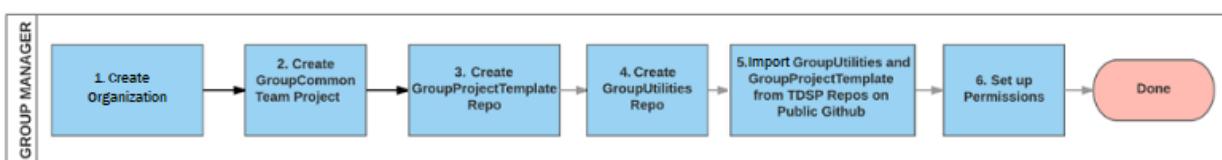
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributor tasks for a data science team](#)

Team Data Science Process group manager tasks

3/10/2022 • 7 minutes to read • [Edit Online](#)

This article describes the tasks that a *group manager* completes for a data science organization. The group manager manages the entire data science unit in an enterprise. A data science unit may have several teams, each of which is working on many data science projects in distinct business verticals. The group manager's objective is to establish a collaborative group environment that standardizes on the [Team Data Science Process](#) (TDSP). For an outline of all the personnel roles and associated tasks handled by a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

The following diagram shows the six main group manager setup tasks. Group managers may delegate their tasks to surrogates, but the tasks associated with the role don't change.



1. Set up an [Azure DevOps organization](#) for the group.
2. Create the default **GroupCommon** project in the Azure DevOps organization.
3. Create the **GroupProjectTemplate** repository in Azure Repos.
4. Create the **GroupUtilities** repository in Azure Repos.
5. Import the contents of the Microsoft TDSP team's **ProjectTemplate** and **Utilities** repositories into the group common repositories.
6. Set up **membership** and **permissions** for team members to access the group.

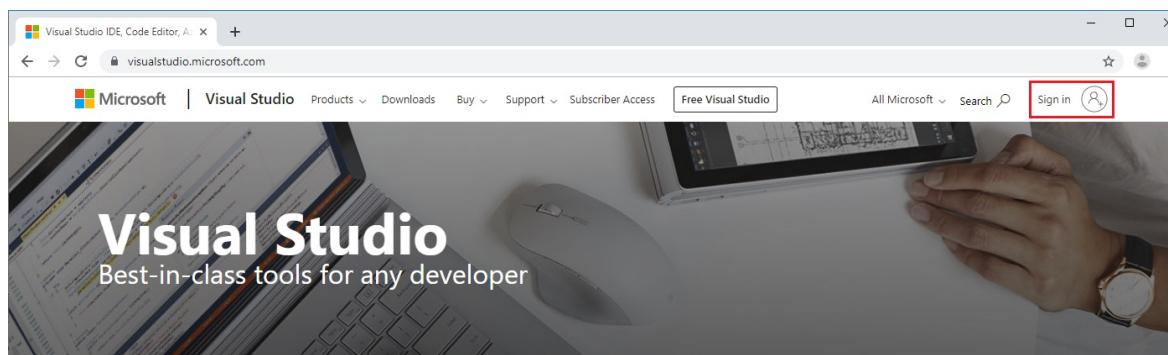
The following tutorial walks through the steps in detail.

NOTE

This article uses Azure DevOps to set up a TDSP group environment, because that is how to implement TDSP at Microsoft. If your group uses other code hosting or development platforms, the Group Manager's tasks are the same, but the way to complete them may be different.

Create an organization and project in Azure DevOps

1. Go to visualstudio.microsoft.com, select **Sign in** at upper right, and sign into your Microsoft account.



If you don't have a Microsoft account, select **Sign up now**, create a Microsoft account, and sign in using this account. If your organization has a Visual Studio subscription, sign in with the credentials for that

subscription.

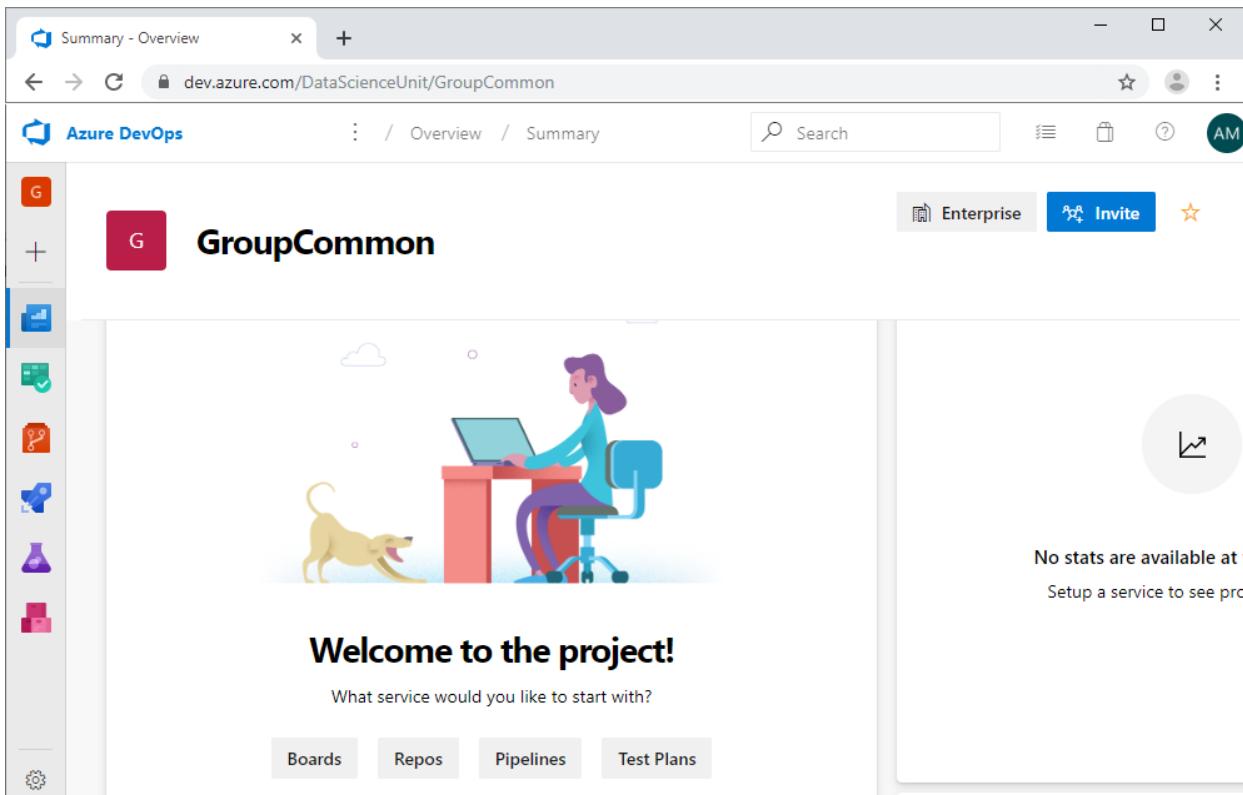
2. After you sign in, at upper right on the Azure DevOps page, select **Create new organization**.

The screenshot shows the Azure DevOps Organizations page. On the left, there is a sidebar for a user named 'A Group Manager' with a profile picture containing 'AM', an email address (gm@fabrikam.com), and a location (United States). The main area displays 'Azure DevOps Organizations' and a list of projects: 'TDSP Customer Project', 'FabrikamFiber', and 'microprofile1'. A red box highlights the 'Create new organization' button in the top right corner of the main content area.

3. If you're prompted to agree to the Terms of Service, Privacy Statement, and Code of Conduct, select **Continue**.
4. In the signup dialog, name your Azure DevOps organization and accept the host region assignment, or drop down and select a different region. Then select **Continue**.
5. Under **Create a project to get started**, enter *GroupCommon*, and then select **Create project**.

The screenshot shows the 'Create a project to get started' dialog. On the left, there is a sidebar with organization names: 'DataScienceUnit' (selected), 'mseng', 'ceapex', and 'fabrikam-org', along with links for '8 more organizations' and 'New organization'. The main area has a title 'Create a project to get started' and a 'Project name *' field containing 'GroupCommon'. Below it, there is a 'Visibility' section with three options: 'Public' (Anyone on the internet can view the project. Certain features like TFVC are not supported.), 'Enterprise' (Members of your enterprise can view the project. This is selected and highlighted with a red box), and 'Private' (Only people you give access to will be able to view this project.). A red box also highlights the '+ Create project' button at the bottom.

The **GroupCommon** project **Summary** page opens. The page URL is <https://<servername>/<organization-name>/GroupCommon>.



Set up the group common repositories

Azure Repos hosts the following types of repositories for your group:

- **Group common repositories:** General-purpose repositories that multiple teams within a data science unit can adopt for many data science projects.
- **Team repositories:** Repositories for specific teams within a data science unit. These repositories are specific for a team's needs, and may be used for multiple projects within that team, but are not general enough to be used across multiple teams within a data science unit.
- **Project repositories:** Repositories for specific projects. Such repositories may not be general enough for multiple projects within a team, or for other teams in a data science unit.

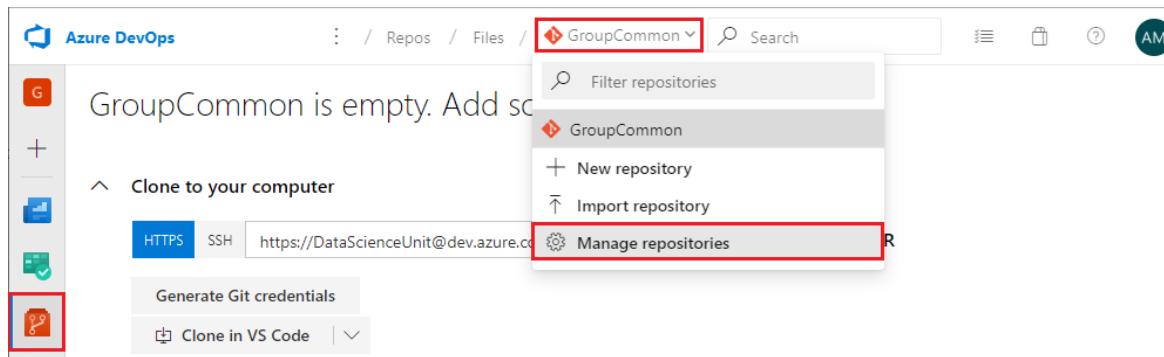
To set up the group common repositories in your project, you:

- Rename the default **GroupCommon** repository to **GroupProjectTemplate**
- Create a new **GroupUtilities** repository

Rename the default project repository to **GroupProjectTemplate**

To rename the default **GroupCommon** project repository to **GroupProjectTemplate**:

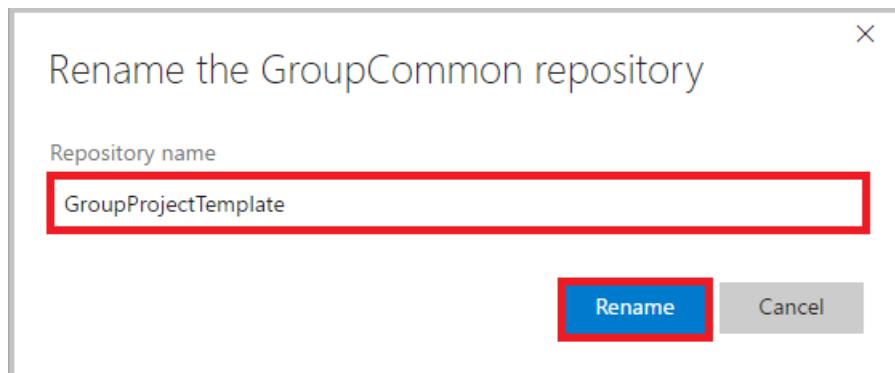
1. On the **GroupCommon** project **Summary** page, select **Repos**. This action takes you to the default **GroupCommon** repository of the **GroupCommon** project, which is currently empty.
2. At the top of the page, drop down the arrow next to **GroupCommon** and select **Manage repositories**.



3. On the **Project Settings** page, select the ... next to **GroupCommon**, and then select **Rename repository**.

The screenshot shows the 'Repositories' section of the 'GroupCommon' project settings. The 'GroupCommon' repository is selected. A context menu is open with the 'Rename repository...' option highlighted. The 'Security' tab is active, displaying a detailed list of users and their permissions.

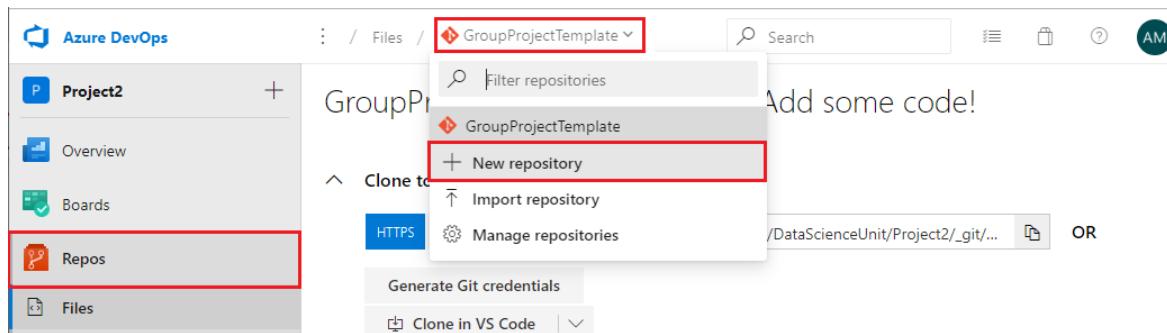
4. In the **Rename the GroupCommon repository** popup, enter *GroupProjectTemplate*, and then select **Rename**.



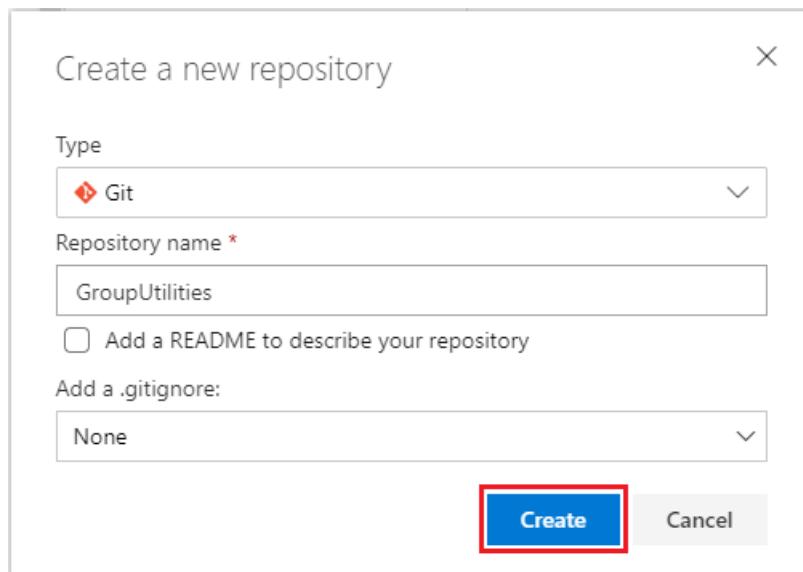
Create the GroupUtilities repository

To create the **GroupUtilities** repository:

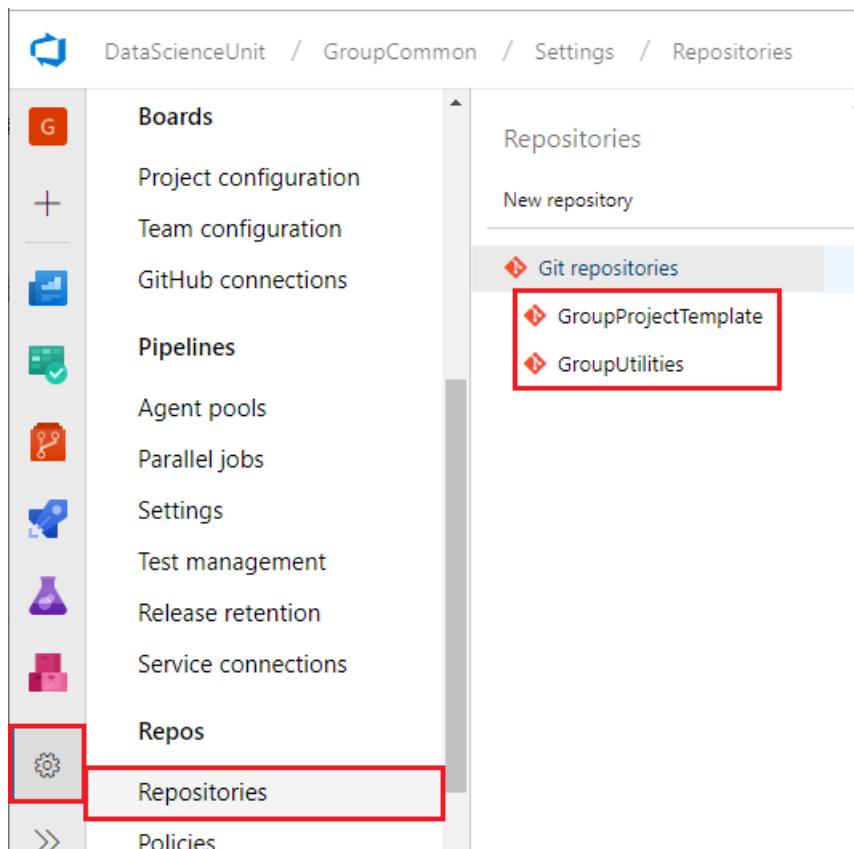
1. On the **GroupCommon** project **Summary** page, select **Repos**.
2. At the top of the page, drop down the arrow next to **GroupProjectTemplate** and select **New repository**.



3. In the **Create a new repository** dialog, select **Git** as the **Type**, enter *GroupUtilities* as the **Repository name**, and then select **Create**.



4. On the **Project Settings** page, select **Repositories** under **Repos** in the left navigation to see the two group repositories: **GroupProjectTemplate** and **GroupUtilities**.



Import the Microsoft TDSP team repositories

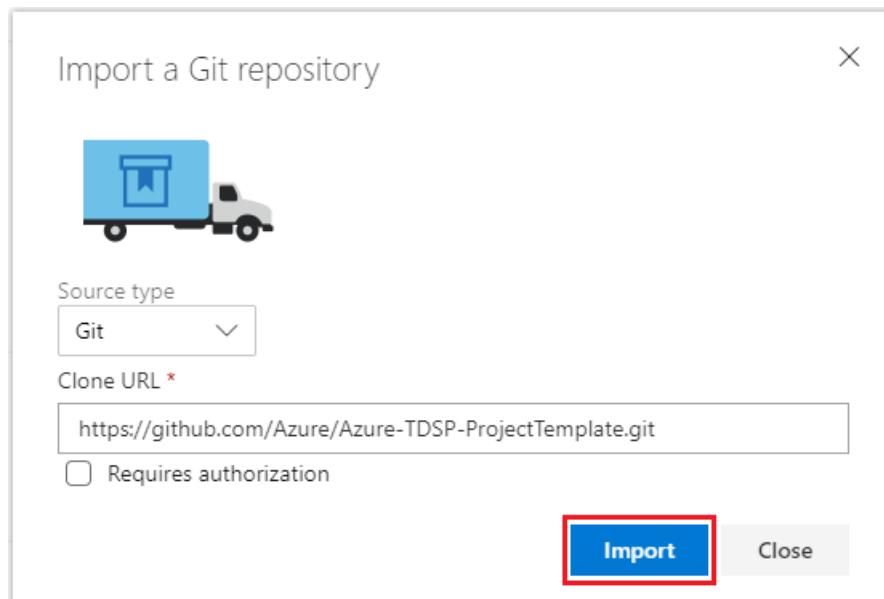
In this part of the tutorial, you import the contents of the **ProjectTemplate** and **Utilities** repositories managed by the Microsoft TDSP team into your **GroupProjectTemplate** and **GroupUtilities** repositories.

To import the TDSP team repositories:

1. From the **GroupCommon** project home page, select **Repos** in the left navigation. The default **GroupProjectTemplate** repo opens.
2. On the **GroupProjectTemplate is empty** page, select **Import**.

The screenshot shows the Azure DevOps interface for the **GroupCommon** project. The left sidebar is visible with options like Overview, Boards, and Repos. The main content area displays the message "GroupProjectTemplate is empty. Add some code!" and provides instructions for cloning the repository. It shows two cloning methods: HTTPS and SSH, with the HTTPS URL being https://DataScienceUnit@dev.azure.com/DataScienceUnit/GroupCommon/_git/GroupProjectTemplate. Below the cloning instructions, there's a section for importing a repository, which includes a red box around the "Import" button.

3. In the **Import a Git repository** dialog, select **Git** as the **Source type**, and enter <https://github.com/Azure/Azure-TDSP-ProjectTemplate.git> for the **Clone URL**. Then select **Import**. The contents of the Microsoft TDSP team ProjectTemplate repository are imported into your **GroupProjectTemplate** repository.



4. At the top of the **Repos** page, drop down and select the **GroupUtilities** repository.

Each of your two group repositories now contains all the files, except those in the `.git` directory, from the Microsoft TDSP team's corresponding repository.

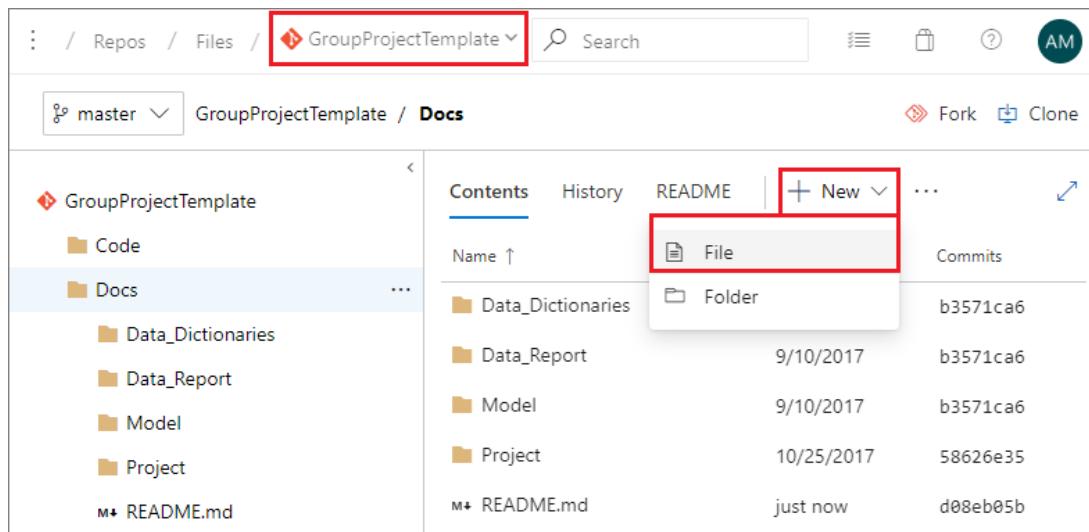
Customize the contents of the group repositories

If you want to customize the contents of your group repositories to meet the specific needs of your group, you can do that now. You can modify the files, change the directory structure, or add files that your group has developed or that are helpful for your group.

Make changes in Azure Repos

To customize repository contents:

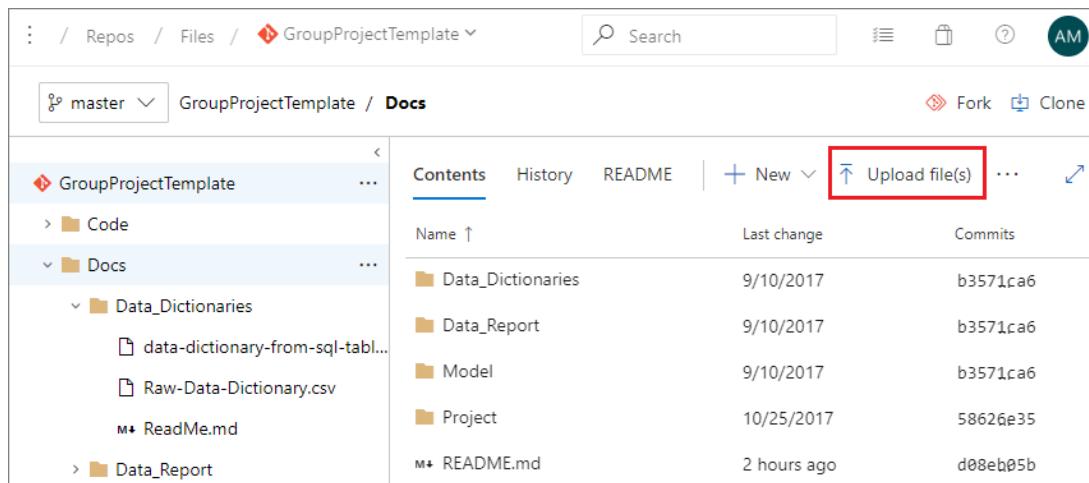
1. On the **GroupCommon** project **Summary** page, select **Repos**.
2. At the top of the page, select the repository you want to customize.
3. In the repo directory structure, navigate to the folder or file you want to change.
 - To create new folders or files, select the arrow next to **New**.



The screenshot shows the Azure Repos interface for the 'GroupProjectTemplate' repository. The left sidebar shows the directory structure: 'Code', 'Docs' (which is expanded to show 'Data_Dictionaries', 'Data_Report', 'Model', 'Project', and 'README.md'), and 'README'. The main pane displays a table of files and folders under the 'Docs' folder. The 'New' button in the top right of the table header is highlighted with a red box. A dropdown menu is open, showing 'File' and 'Folder' options, also highlighted with a red box.

Name ↑	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	just now	d08eb05b

- To upload files, select **Upload file(s)**.



The screenshot shows the Azure Repos interface for the 'GroupProjectTemplate' repository. The left sidebar shows the directory structure: 'Code', 'Docs' (which is expanded to show 'Data_Dictionaries' (containing 'data-dictionary-from-sql-tabl...', 'Raw-Data-Dictionary.csv', and 'ReadMe.md'), and 'Data_Report'). The main pane displays a table of files and folders under the 'Docs' folder. The 'Upload file(s)' button in the top right of the table header is highlighted with a red box.

Name ↑	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	2 hours ago	d08eb05b

- To edit existing files, navigate to the file and then select **Edit**.

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a tree view of files and folders: GroupProjectTemplate (Code, Docs, Data_Dictionaries, data-dictionary-from-sql-table..., Raw-Data-Dictionary.csv, ReadMe.md, Data_Report, Model, Project). The main content area shows the README.md file with the following text:

Folder for hosting all documents for the Data Science Unit

Documents will contain information about the following

1. System architecture
2. Data dictionaries
3. Reports related to data understanding, modeling
4. Project management and planning docs
5. Information obtained from a business owner or client about the project
6. Docs and presentations prepared to share information about the project

4. After adding or editing files, select **Commit**.

The screenshot shows a 'Commit' dialog box overlaid on a GitHub repository page. The dialog box has the following fields:

- Comment:** Updated README.md
- Branch name:** master
- Work items to link:** Search work items by ID or title

At the bottom right of the dialog box, the **Commit** button is highlighted with a red box.

Make changes using your local machine or DSVM

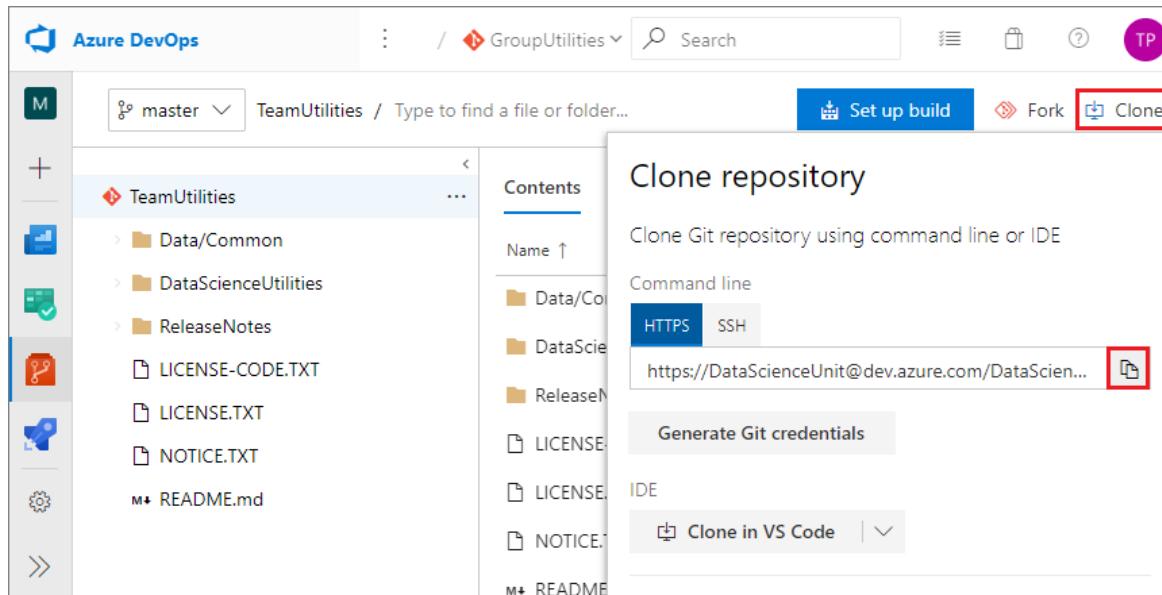
If you want to make changes using your local machine or DSVM and push the changes up to the group repositories, make sure you have the prerequisites for working with Git and DSVMs:

- An Azure subscription, if you want to create a DSVM.
- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.

- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the [Create SSH public key](#) section in the [Platforms and tools appendix](#).

First, copy or *clone* the repository to your local machine.

- On the **GroupCommon** project **Summary** page, select **Repos**, and at the top of the page, select the repository you want to clone.
- On the repo page, select **Clone** at upper right.
- In the **Clone repository** dialog, select **HTTPS** for an HTTP connection, or **SSH** for an SSH connection, and copy the clone URL under **Command line** to your clipboard.



- On your local machine, create the following directories:

- For Windows: `C:\GitRepos\GroupCommon`
- For Linux, `$/GitRepos/GroupCommon` on your home directory

- Change to the directory you created.

- In Git Bash, run the command `git clone <clone URL>`.

For example, either of the following commands clones the **GroupUtilities** repository to the **GroupCommon** directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/_git/GroupUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/GroupCommon/GroupUtilities
```

After making whatever changes you want in the local clone of your repository, you can push the changes to the shared group common repositories.

Run the following Git Bash commands from your local **GroupProjectTemplate** or **GroupUtilities** directory.

```
git add .
git commit -m "push from local"
git push
```

NOTE

If this is the first time you commit to a Git repository, you may need to configure global parameters `user.name` and `user.email` before you run the `git commit` command. Run the following two commands:

```
git config --global user.name <your name>
```

```
git config --global user.email <your email address>
```

If you're committing to several Git repositories, use the same name and email address for all of them. Using the same name and email address is convenient when building Power BI dashboards to track your Git activities in multiple repositories.

Add group members and configure permissions

To add members to the group:

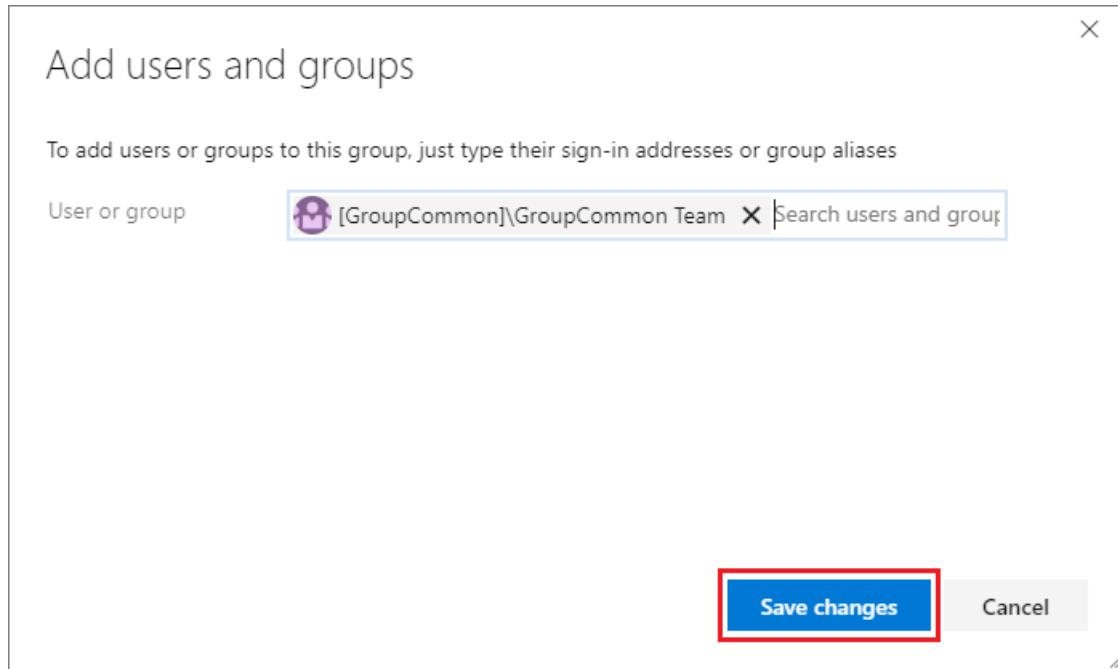
1. In Azure DevOps, from the **GroupCommon** project home page, select **Project settings** from the left navigation.
2. From the **Project Settings** left navigation, select **Teams**, then on the **Teams** page, select the **GroupCommon Team**.

The screenshot shows the Azure DevOps interface for managing a project. On the left, there's a vertical sidebar with various icons and sections. The 'Groups' icon (a person icon) is highlighted with a red box. Below it, the 'Teams' section is also highlighted with a red box. The main content area is titled 'Teams' and shows a table with one row. The row for 'GroupCommon Team' has a profile picture, the name 'GroupCommon Team', 1 member, and the description 'The default project team.' A 'New team' button and a refresh icon are also visible in the table header.

3. On the **Team Profile** page, select **Add**.

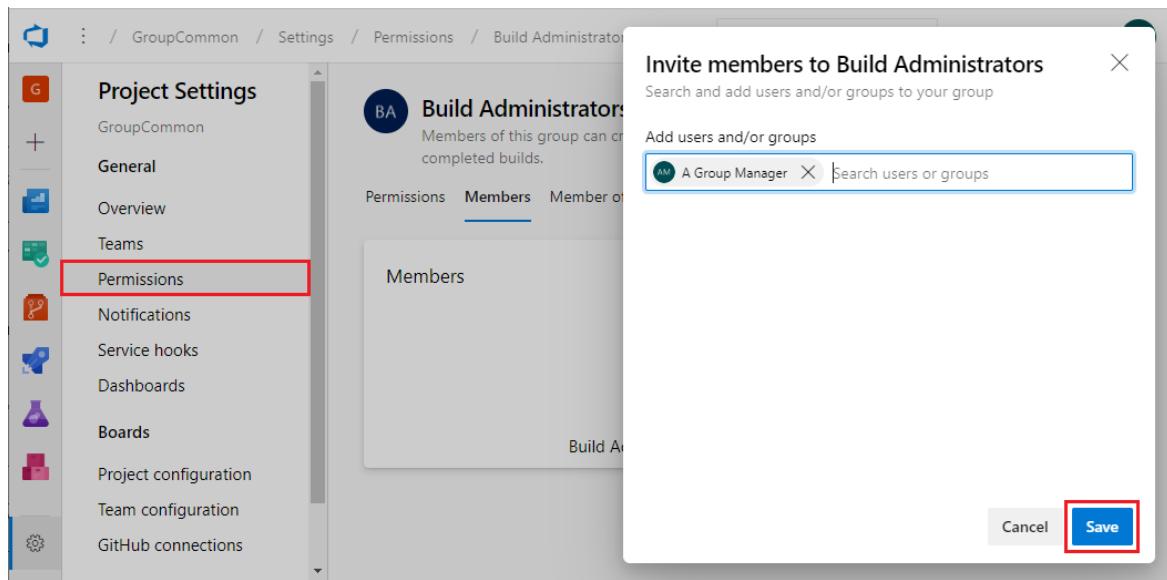
<p>Team Profile</p>  <p>Name GroupCommon Team</p> <p>Description The default project team.</p>	<p>GroupCommon Team</p> <p>Members</p> <table border="1"> <thead> <tr> <th>Display Name</th> <th>Username Or Scope</th> </tr> </thead> <tbody> <tr> <td>AM A Group Manager</td> <td>gm@fabrikam.com</td> </tr> </tbody> </table> <p>membership direct</p>	Display Name	Username Or Scope	AM A Group Manager	gm@fabrikam.com
Display Name	Username Or Scope				
AM A Group Manager	gm@fabrikam.com				

4. In the **Add users and groups** dialog, search for and select members to add to the group, and then select **Save changes**.



To configure permissions for members:

1. From the **Project Settings** left navigation, select **Permissions**.
2. On the **Permissions** page, select the group you want to add members to.
3. On the page for that group, select **Members**, and then select **Add**.
4. In the **Invite members** popup, search for and select members to add to the group, and then select **Save**.



Next steps

Here are links to detailed descriptions of the other roles and tasks in the Team Data Science Process:

- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributor tasks for a data science team](#)

Tasks for the team lead on a Team Data Science Process team

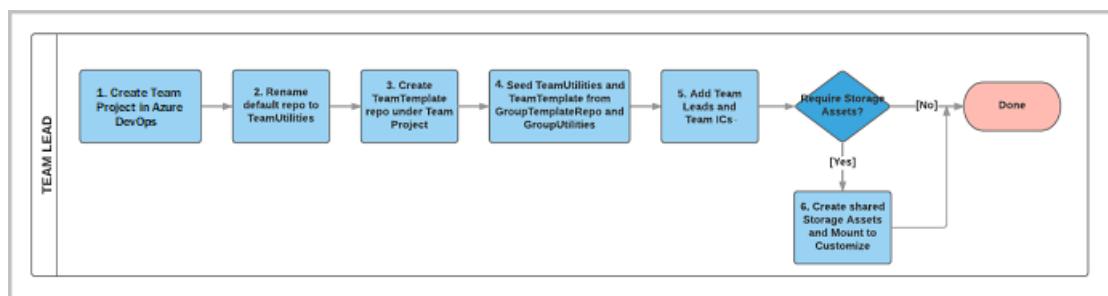
3/10/2022 • 11 minutes to read • [Edit Online](#)

This article describes the tasks that a *team lead* completes for their data science team. The team lead's objective is to establish a collaborative team environment that standardizes on the [Team Data Science Process](#) (TDSP). The TDSP is designed to help improve collaboration and team learning.

The TDSP is an agile, iterative data science methodology to efficiently deliver predictive analytics solutions and intelligent applications. The process distills the best practices and structures from Microsoft and the industry. The goal is successful implementation of data science initiatives and fully realizing the benefits of their analytics programs. For an outline of the personnel roles and associated tasks for a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

A team lead manages a team consisting of several data scientists in the data science unit of an enterprise. Depending on the data science unit's size and structure, the [group manager](#) and the team lead might be the same person, or they could delegate their tasks to surrogates. But the tasks themselves do not change.

The following diagram shows the workflow for the tasks the team lead completes to set up a team environment:



1. Create a **team project** in the group's organization in Azure DevOps.
2. Rename the default team repository to **TeamUtilities**.
3. Create a new **TeamTemplate** repository in the team project.
4. Import the contents of the group's **GroupUtilities** and **GroupProjectTemplate** repositories into the **TeamUtilities** and **TeamTemplate** repositories.
5. Set up **security control** by adding team members and configuring their permissions.
6. If required, create team data and analytics resources:
 - Add team-specific utilities to the **TeamUtilities** repository.
 - Create **Azure file storage** to store data assets that can be useful for the entire team.
 - Mount the Azure file storage to the team lead's **Data Science Virtual Machine** (DSVM) and add data assets to it.

The following tutorial walks through the steps in detail.

NOTE

This article uses Azure DevOps and a DSVM to set up a TDSP team environment, because that is how to implement TDSP at Microsoft. If your team uses other code hosting or development platforms, the team lead tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that the following resources and permissions have been set up by your [group manager](#):

- The Azure DevOps **organization** for your data unit
- **GroupProjectTemplate** and **GroupUtilities** repositories, populated with the contents of the Microsoft TDSP team's **ProjectTemplate** and **Utilities** repositories
- Permissions on your organization account for you to create projects and repositories for your team

To be able to clone repositories and modify their content on your local machine or DSVM, or set up Azure file storage and mount it to your DSVM, you need the following:

- An Azure subscription.
- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

Create a team project and repositories

In this section, you create the following resources in your group's Azure DevOps organization:

- The **MyTeam** project in Azure DevOps
- The **TeamTemplate** repository
- The **TeamUtilities** repository

The names specified for the repositories and directories in this tutorial assume that you want to establish a separate project for your own team within your larger data science organization. However, the entire group can choose to work under a single project created by the group manager or organization administrator. Then, all the data science teams create repositories under this single project. This scenario might be valid for:

- A small data science group that doesn't have multiple data science teams.
- A larger data science group with multiple data science teams that nevertheless wants to optimize inter-team collaboration with activities such as group-level sprint planning.

If teams choose to have their team-specific repositories under a single group project, the team leads should create the repositories with names like *<TeamName>Template* and *<TeamName>Utilities*. For instance: *TeamATemplate* and *TeamAUtilities*.

In any case, team leads need to let their team members know which template and utilities repositories to set up and clone. Project leads should follow the [project lead tasks for a data science team](#) to create project repositories, whether under separate projects or a single project.

Create the MyTeam project

To create a separate project for your team:

1. In your web browser, go to your group's Azure DevOps organization home page at URL <https://<server name>/<organization name>>, and select **New project**.

The screenshot shows the Azure DevOps organization home page for 'DataScienceUnit'. On the right side, there is a prominent blue button labeled '+ New project' with a red box drawn around it. The main area displays a project card for 'GroupCommon'.

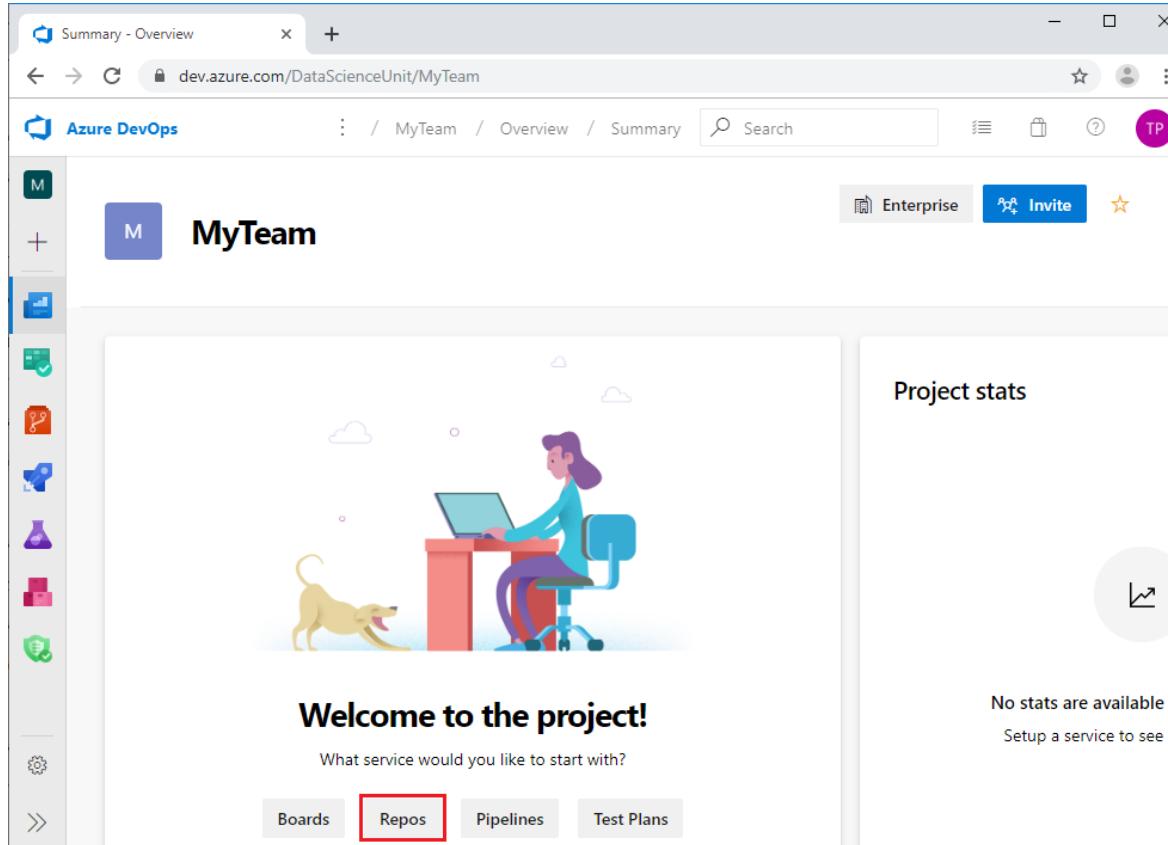
2. In the **Create project** dialog, enter your team name, such as *MyTeam*, under **Project name**, and then select **Advanced**.
3. Under **Version control**, select **Git**, and under **Work item process**, select **Agile**. Then select **Create**.

The screenshot shows the 'Create new project' dialog. The 'Project name' field contains 'MyTeam' with a red box around it. The 'Visibility' section shows 'Enterprise' selected (radio button highlighted with a blue box). The 'Advanced' button at the bottom of the visibility section has a red box around it. The 'Version control' dropdown is set to 'Git' (highlighted with a red box) and the 'Work item process' dropdown is set to 'Agile' (highlighted with a red box). The 'Create' button at the bottom right is also highlighted with a red box.

The team project **Summary** page opens, with page URL <https://<server name>/<organization name>/<team name>>.

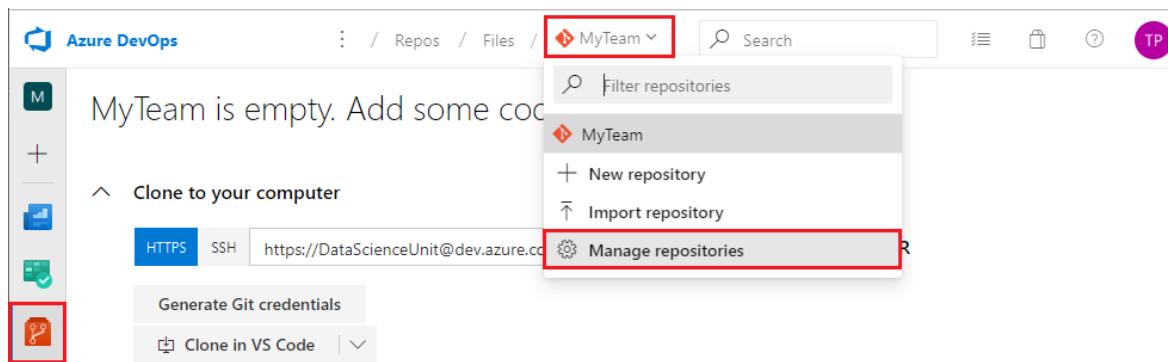
Rename the MyTeam default repository to TeamUtilities

1. On the **MyTeam** project **Summary** page, under **What service would you like to start with?**, select **Repos**.



The screenshot shows the Azure DevOps interface for the 'MyTeam' project. The top navigation bar includes 'Enterprise' and 'Invite' buttons. The main area features a cartoon illustration of a person working at a desk with a dog. Below the illustration, the text 'Welcome to the project!' is displayed, followed by a question 'What service would you like to start with?'. Underneath, there are four tabs: 'Boards', 'Repos' (which is highlighted with a red box), 'Pipelines', and 'Test Plans'. To the left, a vertical sidebar contains icons for Boards, Repositories, Pipelines, Test Plans, and Project Settings.

2. On the **MyTeam** repo page, select the **MyTeam** repository at the top of the page, and then select **Manage repositories** from the dropdown.



The screenshot shows the Azure DevOps interface for the 'MyTeam' repository. The top navigation bar includes 'Enterprise' and 'Invite' buttons. The main area displays the message 'MyTeam is empty. Add some code...'. Below this, there's a 'Clone to your computer' section with options for 'HTTPS' and 'SSH'. A red box highlights the 'Manage repositories' button in a dropdown menu that appears when the repository name 'MyTeam' is selected. Other options in the dropdown include 'New repository' and 'Import repository'. To the left, a vertical sidebar contains icons for Boards, Repositories (highlighted with a red box), Pipelines, Test Plans, and Project Settings.

3. On the **Project Settings** page, select the ... next to the **MyTeam** repository, and then select **Rename repository**.

The screenshot shows the 'Project Settings' page for a project named 'MyTeam'. On the left, there's a sidebar with various settings options like General, Overview, Teams, Permissions, Notifications, Service hooks, Dashboards, Boards, Project configuration, Team configuration, GitHub connections, and Pipelines. The 'Pipelines' icon is highlighted with a red box. The main area shows a list of repositories under 'Git repositories'. One repository, 'MyTeam', is selected and has a red box around its three-dot menu icon. A tooltip 'Rename repository...' is shown next to this icon. To the right, there's a 'Security for MyTeam repository' panel with tabs for Security, Options, and Policies, and a list of security roles.

4. In the **Rename the MyTeam repository** popup, enter *TeamUtilities*, and then select **Rename**.

Create the TeamTemplate repository

1. On the **Project Settings** page, select **New repository**.

The screenshot shows the 'Project Settings' page for a project named 'MyTeam'. The sidebar and repository list are identical to the previous screenshot. However, the 'New repository' button at the top of the repository list is highlighted with a red box. The rest of the interface is the same, showing the security configuration for the repository.

Or, select **Repos** from the left navigation of the **MyTeam** project **Summary** page, select a repository at the top of the page, and then select **New repository** from the dropdown.

2. In the **Create a new repository** dialog, make sure **Git** is selected under **Type**. Enter *TeamTemplate* under **Repository name**, and then select **Create**.

Create a new repository

Type
Git

Repository name *
TeamTemplate

Add a README to describe your repository

Add a .gitignore:
None

Create **Cancel**

3. Confirm that you can see the two repositories **TeamUtilities** and **TeamTemplate** on your project settings page.

The screenshot shows the 'Project Settings' page for a project named 'MyTeam'. The left sidebar has a red box around the 'Pipelines' icon. The main area shows the 'Repositories' section with 'Git repositories' selected. It lists two repositories: 'TeamTemplate' and 'TeamUtilities'. To the right, there is a 'Security' panel titled 'Security for MyTeam repository' with tabs for 'Security', 'Options', and 'Policies'. It shows a list of security groups: 'Azure DevOps Groups', 'Project Collection Administrators', 'Project Collection Build Service Accounts', 'Project Collection Service Accounts', 'Build Administrators', 'Contributors', 'Project Administrators', and 'Readers'. A search bar is also present in the security panel.

Import the contents of the group common repositories

To populate your team repositories with the contents of the group common repositories set up by your group manager:

1. From your **MyTeam** project home page, select **Repos** in the left navigation. If you get a message that the **MyTeam** template is not found, select the link in **Otherwise, navigate to your default TeamTemplate repository.**

The default **TeamTemplate** repository opens.

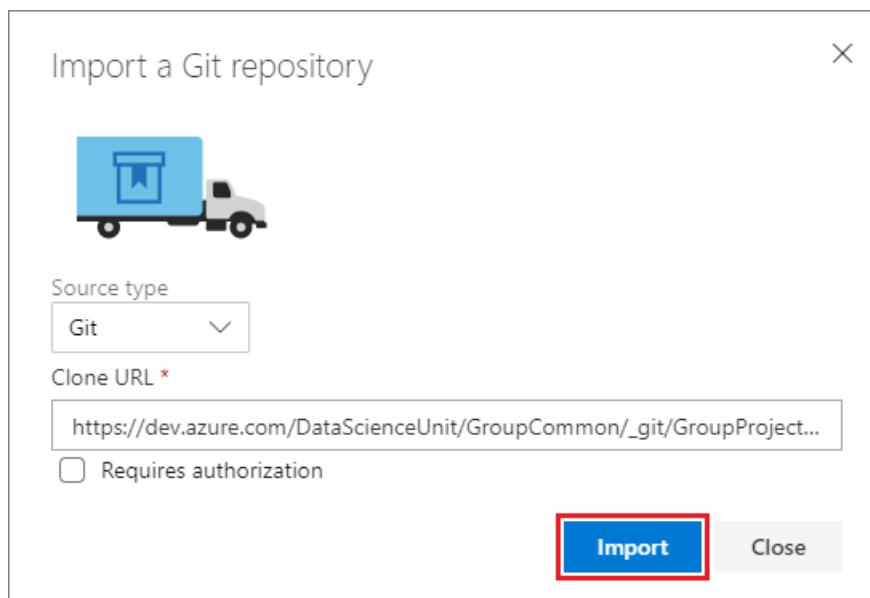
2. On the **TeamTemplate** is empty page, select **Import**.

The screenshot shows the Azure DevOps interface for a repository named 'TeamTemplate'. The left sidebar has a red box around the 'Clone' icon. The main content area displays cloning instructions:

- Clone to your computer**: Shows HTTPS and SSH URLs, a 'Generate Git credentials' button, and a 'Clone in VS Code' dropdown. A note says: "Having problems authenticating in Git? Be sure to get the latest version of Git for Windows or our plugins for IntelliJ, Eclipse, Android Studio or Windows command line."
- or push an existing repository from command line**: Shows a command-line interface with the following text:

```
git remote add origin https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/TeamTemplate
git push -u origin --all
```
- or import a repository**: Contains an 'Import' button.

3. In the **Import a Git repository** dialog, select **Git** as the **Source type**, and enter the URL for your group common template repository under **Clone URL**. The URL is https://<server name>/<organization name>/_git/<repository name>. For example: https://dev.azure.com/DataScienceUnit/GroupCommon/_git/GroupProjectTemplate.
4. Select **Import**. The contents of your group template repository are imported into your team template repository.



5. At the top of your project's **Repos** page, drop down and select the **TeamUtilities** repository.
6. Repeat the import process to import the contents of your group common utilities repository, for example **GroupUtilities**, into your **TeamUtilities** repository.

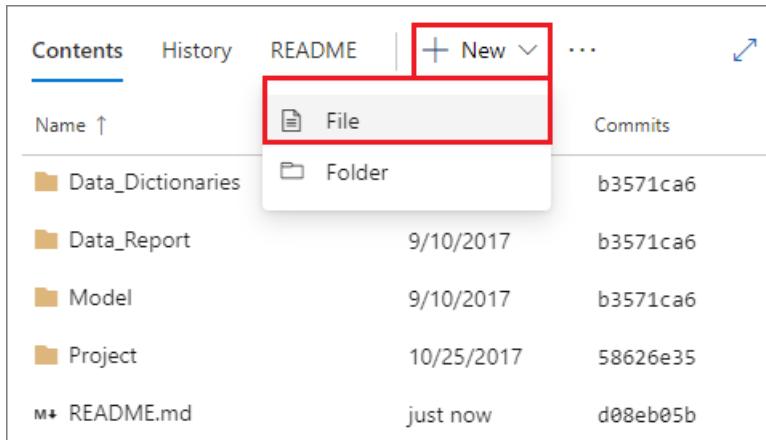
Each of your two team repositories now contains the files from the corresponding group common repository.

Customize the contents of the team repositories

If you want to customize the contents of your team repositories to meet your team's specific needs, you can do that now. You can modify files, change the directory structure, or add files and folders.

To modify, upload, or create files or folders directly in Azure DevOps:

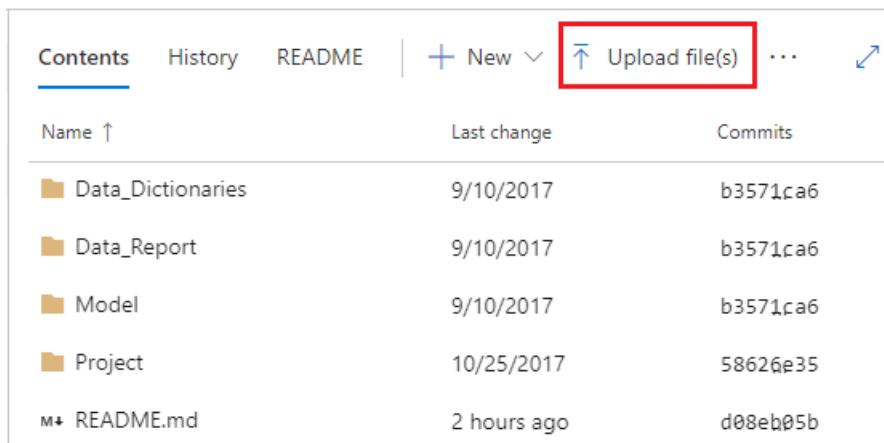
1. On the **MyTeam** project **Summary** page, select **Repos**.
2. At the top of the page, select the repository you want to customize.
3. In the repo directory structure, navigate to the folder or file you want to change.
 - To create new folders or files, select the arrow next to **New**.



The screenshot shows a list of files and folders in a repository. At the top right, there is a 'New' button with a plus sign and a dropdown arrow. A red box highlights this button. A tooltip from the dropdown menu shows two options: 'File' and 'Folder'. The list includes:

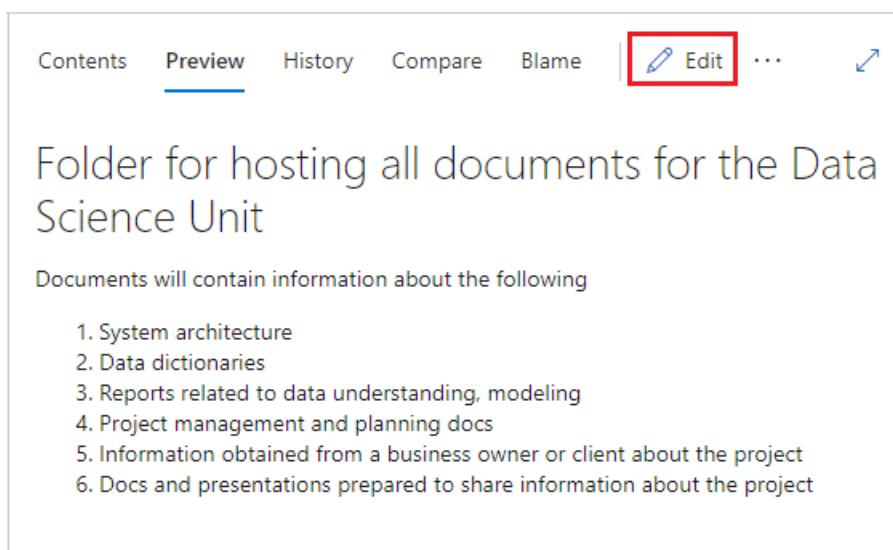
Name	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	just now	d08eb05b

- To upload files, select **Upload file(s)**.



The screenshot shows the same repository interface as above, but the 'New' button has been replaced by an 'Upload file(s)' button with an upward arrow icon. A red box highlights this button. The rest of the interface and data remain the same.

- To edit existing files, navigate to the file and then select **Edit**.



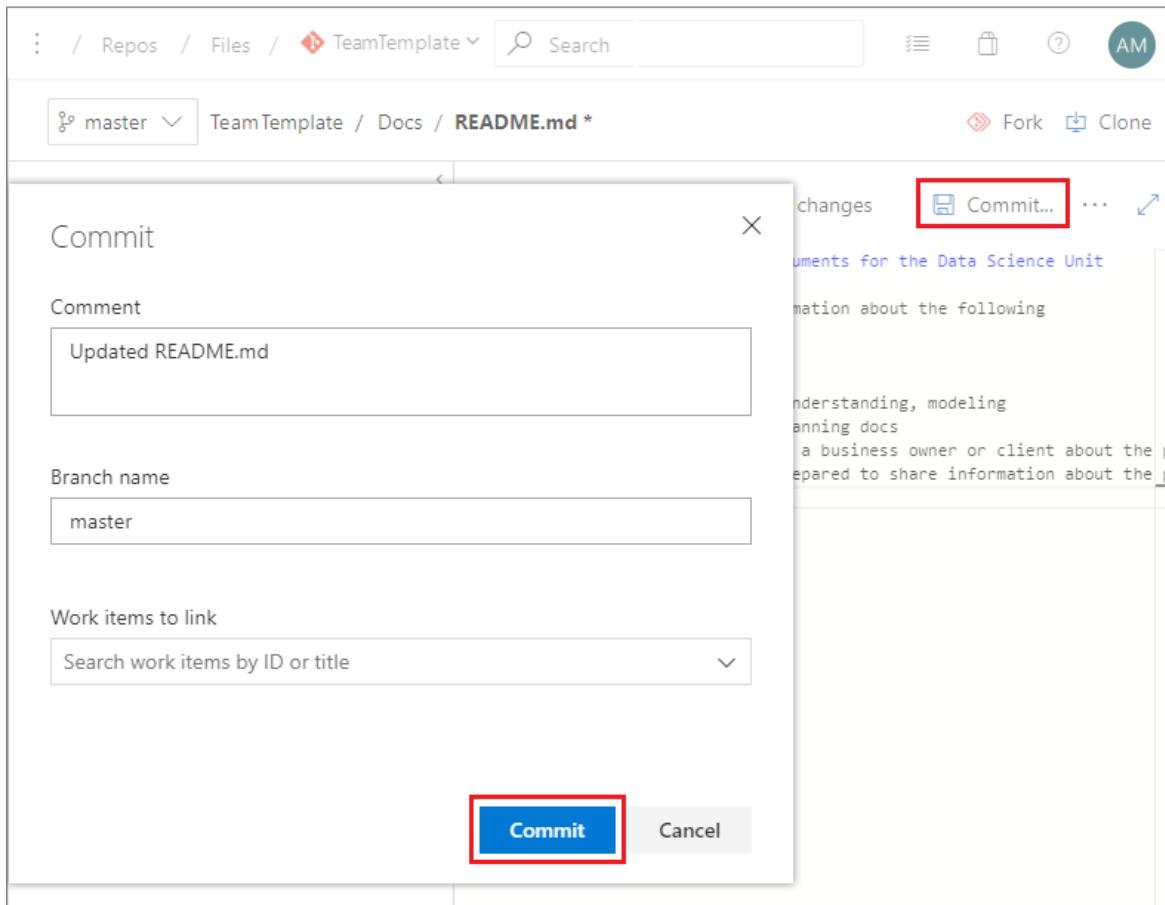
The screenshot shows a detailed view of a file named 'Data_Report'. At the top right, there is an 'Edit' button with a pencil icon. A red box highlights this button. The page contains the following text:

Folder for hosting all documents for the Data Science Unit

Documents will contain information about the following

1. System architecture
2. Data dictionaries
3. Reports related to data understanding, modeling
4. Project management and planning docs
5. Information obtained from a business owner or client about the project
6. Docs and presentations prepared to share information about the project

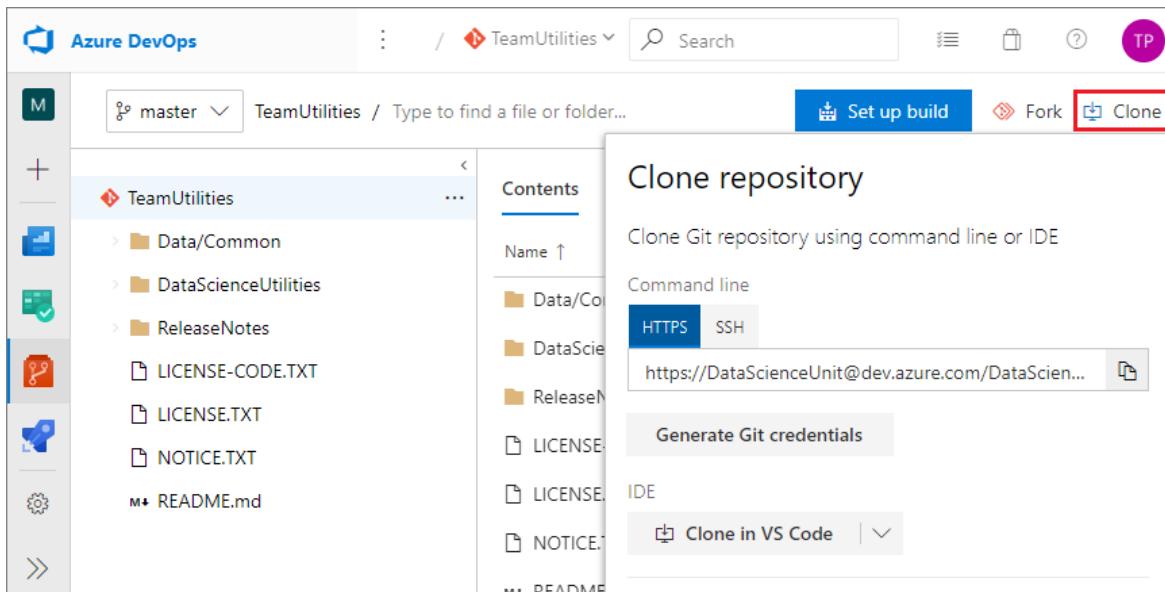
- After adding or editing files, select Commit.



To work with repositories on your local machine or DSVM, you first copy or *clone* the repositories to your local machine, and then commit and push your changes up to the shared team repositories,

To clone repositories:

- On the **MyTeam** project **Summary** page, select **Repos**, and at the top of the page, select the repository you want to clone.
- On the repo page, select **Clone** at upper right.
- In the **Clone repository** dialog, under **Command line**, select **HTTPS** for an HTTP connection or **SSH** for an SSH connection, and copy the clone URL to your clipboard.



4. On your local machine, create the following directories:

- For Windows: C:\GitRepos\MyTeam
- For Linux, \$home/GitRepos/MyTeam

5. Change to the directory you created.

6. In Git Bash, run the command `git clone <clone URL>`, where `<clone URL>` is the URL you copied from the **Clone** dialog.

For example, use one of the following commands to clone the **TeamUtilities** repository to the *MyTeam* directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/TeamUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/MyTeam/TeamUtilities
```

After making whatever changes you want in the local clone of your repository, commit and push the changes to the shared team repositories.

Run the following Git Bash commands from your local **GitRepos\MyTeam\TeamTemplate** or **GitRepos\MyTeam\TeamUtilities** directory.

```
git add .
git commit -m "push from local"
git push
```

NOTE

If this is the first time you commit to a Git repository, you may need to configure global parameters `user.name` and `user.email` before you run the `git commit` command. Run the following two commands:

```
git config --global user.name <your name>
git config --global user.email <your email address>
```

If you're committing to several Git repositories, use the same name and email address for all of them. Using the same name and email address is convenient when building Power BI dashboards to track your Git activities in multiple repositories.

Add team members and configure permissions

To add members to the team:

1. In Azure DevOps, from the **MyTeam** project home page, select **Project settings** from the left navigation.
2. From the **Project Settings** left navigation, select **Teams**, then on the **Teams** page, select the **MyTeam Team**.

The screenshot shows the 'Project Settings' page for 'MyTeam'. The left sidebar has a red box around the 'Teams' option. The main area shows a table with one team entry:

Team Name	Members	Description
MyTeam Team	1	The default project team.

3. On the **Team Profile** page, select Add.

The screenshot shows the 'Team Profile' page for 'MyTeam Team'. The 'Members' section has a red box around the '+ Add...' button. The table shows one member:

Display Name	Username Or Scope
A Group Manager	gm@fabrikam.com

4. In the **Add users and groups** dialog, search for and select members to add to the group, and then select **Save changes**.

The screenshot shows the 'Add users and groups' dialog. The search bar contains '[MyTeam]\MyTeam Team'. The 'Save changes' button at the bottom right is highlighted with a red box.

To configure permissions for team members:

1. From the **Project Settings** left navigation, select **Permissions**.
2. On the **Permissions** page, select the group you want to add members to.
3. On the page for that group, select **Members**, and then select **Add**.
4. In the **Invite members** popup, search for and select members to add to the group, and then select **Save**.

The screenshot shows the Microsoft Teams interface for managing project administrators. On the left, there's a sidebar with various settings options like General, Overview, Teams, Permissions (which is highlighted with a red box), Notifications, Service hooks, and Dashboards. The main area is titled 'Project Administrators' and describes the group's permissions. It has tabs for Permissions, Members (which is selected and highlighted with a blue underline), Member of, and Settings. Below the tabs is a search bar labeled 'Search users and groups'. The 'Members' section lists one member: 'Team Lead' (fabrikamfiber4@i) with the email fabrikamfiber4@hotmail.com. To the right of the member list is a large blue 'Add' button, which is also highlighted with a red box.

Create team data and analytics resources

This step is optional, but sharing data and analytics resources with your entire team has performance and cost benefits. Team members can execute their projects on the shared resources, save on budgets, and collaborate more efficiently. You can create Azure file storage and mount it on your DSVM to share with team members.

For information about sharing other resources with your team, such as Azure HDInsight Spark clusters, see [Platforms and tools](#). That topic provides guidance from a data science perspective on selecting resources that are appropriate for your needs, and links to product pages and other relevant and useful tutorials.

NOTE

To avoid transmitting data across data centers, which might be slow and costly, make sure that your Azure resource group, storage account, and DSVM are all hosted in the same Azure region.

Create Azure file storage

1. Run the following script to create Azure file storage for data assets that are useful for your entire team. The script prompts you for your Azure subscription information, so have that ready to enter.

- On a Windows machine, run the script from the PowerShell command prompt:

```
 wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/TDSP/CreateFileShare.ps1" -outfile "CreateFileShare.ps1"  
.\\CreateFileShare.ps1
```

- On a Linux machine, run the script from the Linux shell:

```
 wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/TDSP/CreateFileShare.sh"  
bash CreateFileShare.sh
```

2. Log in to your Microsoft Azure account when prompted, and select the subscription you want to use.
3. Select the storage account to use, or create a new one under your selected subscription. You can use lowercase characters, numbers, and hyphens for the Azure file storage name.
4. To facilitate mounting and sharing the storage, press Enter or enter *Y* to save the Azure file storage information into a text file in your current directory. You can check in this text file to your **TeamTemplate** repository, ideally under **Docs\Dictionary**, so all projects in your team can access it. You also need the file information to mount your Azure file storage to your Azure DSVM in the next section.

Mount Azure file storage on your local machine or DSVM

1. To mount your Azure file storage to your local machine or DSVM, use the following script.

- On a Windows machine, run the script from the PowerShell command prompt:

```
wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/AttachFileShare.ps1" -outfile "AttachFileShare.ps1"
.\AttachFileShare.ps1
```

- On a Linux machine, run the script from the Linux shell:

```
wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/AttachFileShare.sh"
bash AttachFileShare.sh
```

2. Press Enter or enter *Y* to continue, if you saved an Azure file storage information file in the previous step.

Enter the complete path and name of the file you created.

If you don't have an Azure file storage information file, enter *n*, and follow the instructions to enter your subscription, Azure storage account, and Azure file storage information.

3. Enter the name of a local or TDSP drive to mount the file share on. The screen displays a list of existing drive names. Provide a drive name that doesn't already exist.

4. Confirm that the new drive and storage is successfully mounted on your machine.

Next steps

Here are links to detailed descriptions of the other roles and tasks defined by the Team Data Science Process:

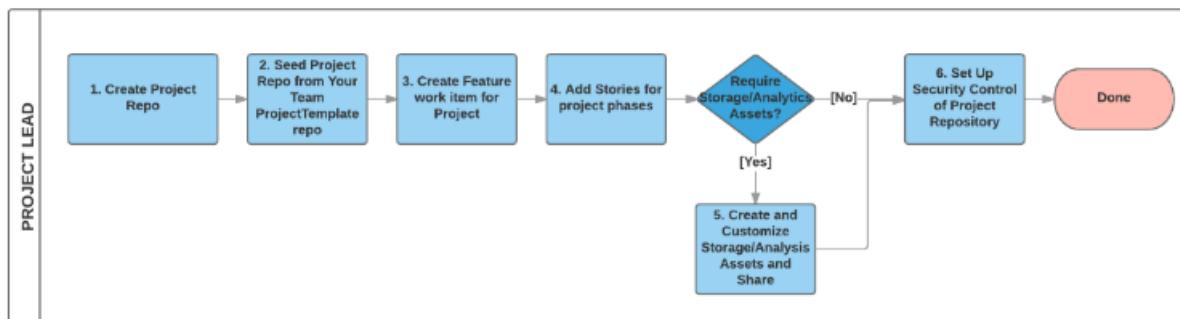
- [Group Manager tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributor tasks for a data science team](#)

Project lead tasks in the Team Data Science Process

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article describes tasks that a *project lead* completes to set up a repository for their project team in the [Team Data Science Process](#) (TDSP). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities to efficiently execute cloud-based, predictive analytics solutions. The TDSP is designed to help improve collaboration and team learning. For an outline of the personnel roles and associated tasks for a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

A project lead manages the daily activities of individual data scientists on a specific data science project in the TDSP. The following diagram shows the workflow for project lead tasks:



This tutorial covers Step 1: Create project repository, and Step 2: Seed project repository from your team ProjectTemplate repository.

For Step 3: Create Feature work item for project, and Step 4: Add Stories for project phases, see [Agile development of data science projects](#).

For Step 5: Create and customize storage/analysis assets and share, if necessary, see [Create team data and analytics resources](#).

For Step 6: Set up security control of project repository, see [Add team members and configure permissions](#).

NOTE

This article uses Azure Repos to set up a TDSP project, because that is how to implement TDSP at Microsoft. If your team uses another code hosting platform, the project lead tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that your [group manager](#) and [team lead](#) have set up the following resources and permissions:

- The Azure DevOps [organization](#) for your data unit
- A team [project](#) for your data science team
- Team template and utilities [repositories](#)
- [Permissions](#) on your organization account for you to create and edit repositories for your project

To clone repositories and modify content on your local machine or Data Science Virtual Machine (DSVM), or set up Azure file storage and mount it to your DSVM, you also need to consider this checklist:

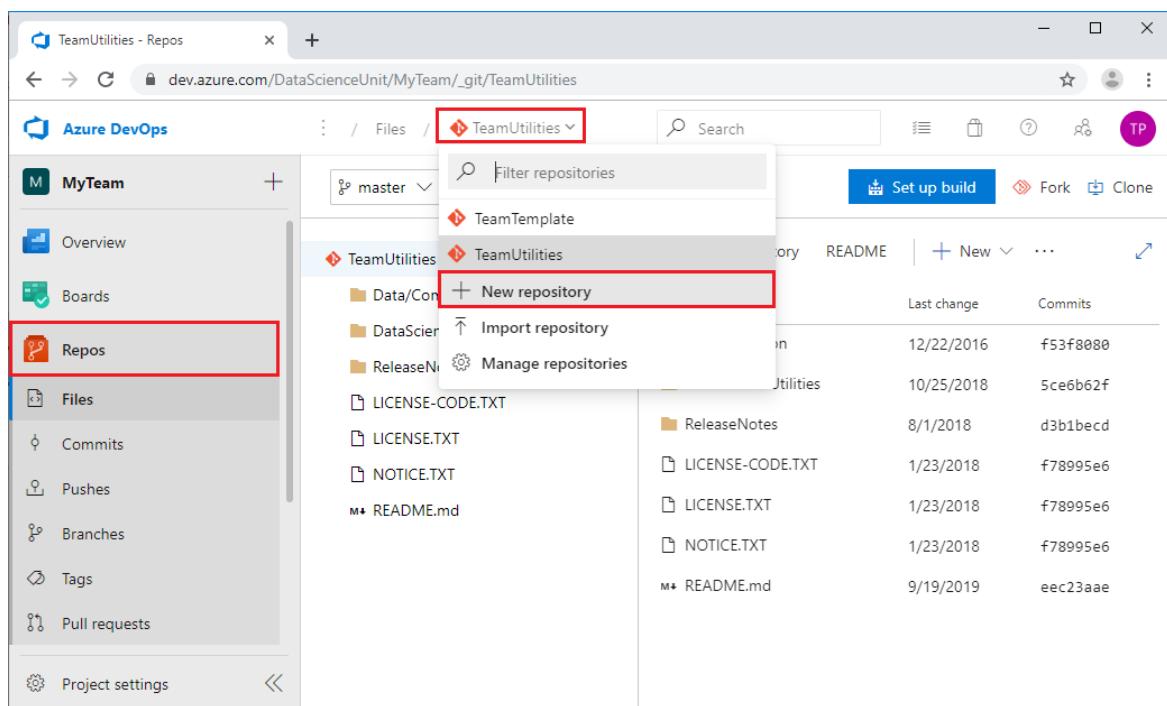
- An Azure subscription.

- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

Create a project repository in your team project

To create a project repository in your team's **MyTeam** project:

1. Go to your team's project **Summary** page at <https://<server name>/<organization name>/<team name>>, for example, <https://dev.azure.com/DataScienceUnit/MyTeam>, and select **Repos** from the left navigation.
2. Select the repository name at the top of the page, and then select **New repository** from the dropdown.



3. In the **Create a new repository** dialog, make sure **Git** is selected under **Type**. Enter *DSProject1* under **Repository name**, and then select **Create**.

Create a new repository

Type
Git

Repository name *

 DSProject1

Add a README to describe your repository

Add a .gitignore:
None

Create Cancel

4. Confirm that you can see the new **DSProject1** repository on your project settings page.

The screenshot shows the Azure DevOps interface for managing repositories. The left sidebar has a 'Repos' section with 'Repositories' selected, indicated by a red box. The main area shows a list of repositories under 'Git repositories'. The 'DSProject1' repository is listed and highlighted with a red box. Other repositories shown are 'TeamTemplate' and 'TeamUtilities'. The top navigation bar shows the project path: DataScienceUnit / MyTeam / Settings / Repositories.

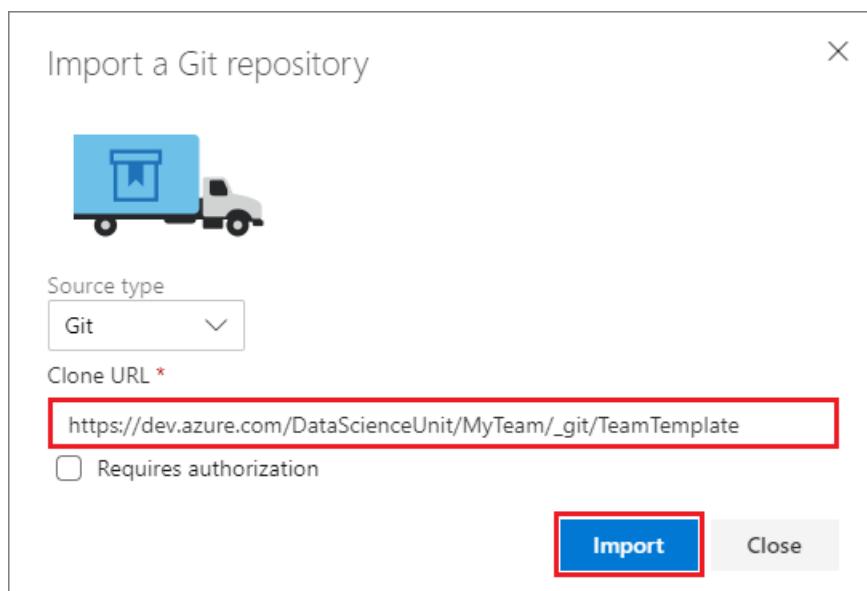
Import the team template into your project repository

To populate your project repository with the contents of your team template repository:

1. From your team's project **Summary** page, select **Repos** in the left navigation.
2. Select the repository name at the top of the page, and select **DSProject1** from the dropdown.
3. On the **DSProject1 is empty** page, select **Import**.

The screenshot shows the Azure DevOps interface for a project named 'DSProject1'. The left sidebar has a 'Git' icon highlighted with a red box. The main content area displays the message 'DSProject1 is empty. Add some code!'. Below this, there's a section titled 'Clone to your computer' with options for 'HTTPS' and 'SSH'. A URL is shown: https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/.... There's also a 'Generate Git credentials' button and a 'Clone in VS Code' dropdown. A note says: 'Having problems authenticating in Git? Be sure to get the latest version of Git for Windows or our plugins for IntelliJ, Eclipse, Android Studio or Windows command line.' Below this is another section titled 'or push an existing repository from command line' with a terminal window showing the commands: 'git remote add origin https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/DSProject1' and 'git push -u origin --all'. At the bottom, there's a section titled 'or import a repository' with a 'Import' button highlighted with a red box.

4. In the **Import a Git repository** dialog, select **Git** as the **Source type**, and enter the URL for your **TeamTemplate** repository under **Clone URL**. The URL is *https://<server name>/<organization name>/<team name>/_git/<team template repository name>*. For example: https://dev.azure.com/DataScienceUnit/MyTeam/_git/TeamTemplate.
5. Select **Import**. The contents of your team template repository are imported into your project repository.



If you need to customize the contents of your project repository to meet your project's specific needs, you can add, delete, or modify repository files and folders. You can work directly in Azure Repos, or clone the repository to your local machine or DSVM, make changes, and commit and push your updates to the shared project repository. Follow the instructions at [Customize the contents of the team repositories](#).

Next steps

Here are links to detailed descriptions of the other roles and tasks defined by the Team Data Science Process:

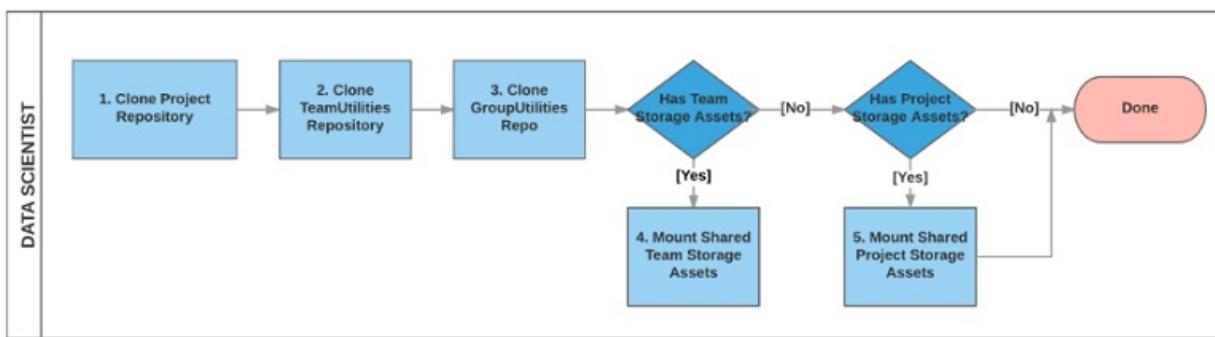
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Individual Contributor tasks for a data science team](#)

Tasks for an individual contributor in the Team Data Science Process

3/10/2022 • 3 minutes to read • [Edit Online](#)

This topic outlines the tasks that an *individual contributor* completes to set up a project in the [Team Data Science Process](#) (TDSP). The objective is to work in a collaborative team environment that standardizes on the TDSP. The TDSP is designed to help improve collaboration and team learning. For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

The following diagram shows the tasks that project individual contributors (data scientists) complete to set up their team environment. For instructions on how to execute a data science project under the TDSP, see [Execution of data science projects](#).



- **ProjectRepository** is the repository your project team maintains to share project templates and assets.
- **TeamUtilities** is the utilities repository your team maintains specifically for your team.
- **GroupUtilities** is the repository your group maintains to share useful utilities across the entire group.

NOTE

This article uses Azure Repos and a Data Science Virtual Machine (DSVM) to set up a TDSP environment, because that is how to implement TDSP at Microsoft. If your team uses other code hosting or development platforms, the individual contributor tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that the following resources and permissions have been set up by your [group manager](#), [team lead](#), and [project lead](#):

- The Azure DevOps **organization** for your data science unit
- A **project repository** set up by your project lead to share project templates and assets
- **GroupUtilities** and **TeamUtilities** repositories set up by the group manager and team lead, if applicable
- Azure **file storage** set up for shared assets for your team or project, if applicable
- **Permissions** for you to clone from and push back to your project repository

To clone repositories and modify content on your local machine or DSVM, or mount Azure file storage to your DSVM, you need to consider this checklist:

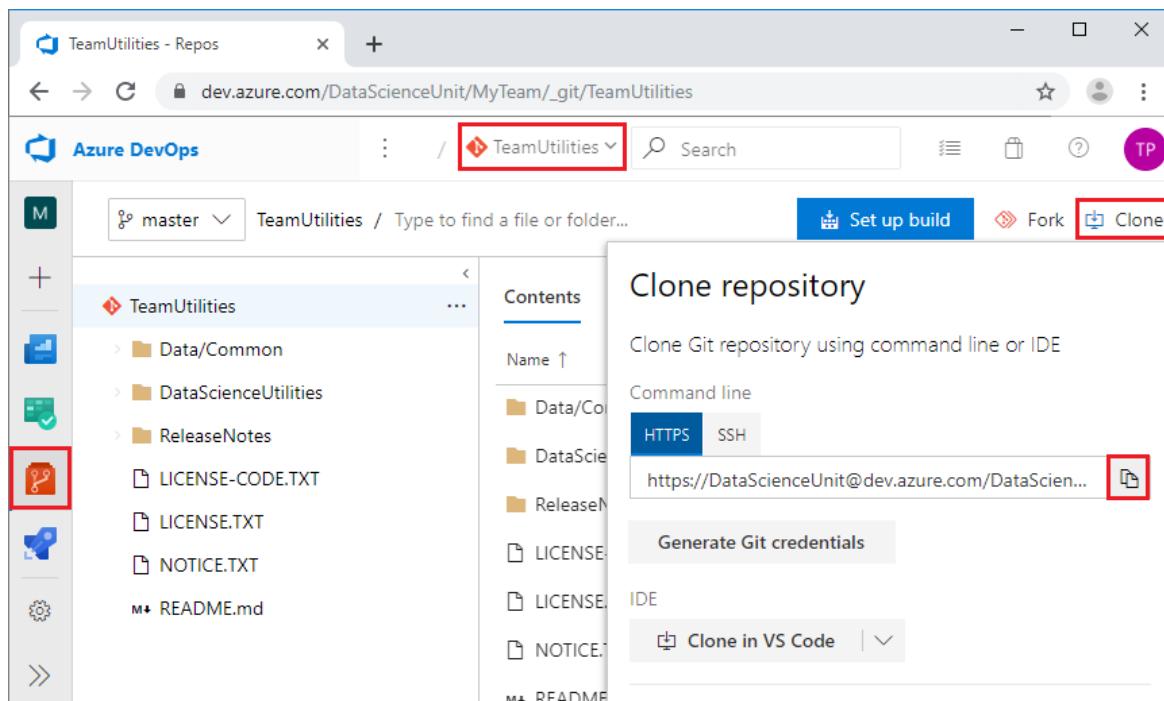
- An Azure subscription.

- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the **Create SSH public key** section in the [Platforms and tools appendix](#).
- The Azure file storage information for any Azure file storage you need to mount to your DSVM.

Clone repositories

To work with repositories locally and push your changes up to the shared team and project repositories, you first copy or *clone* the repositories to your local machine.

1. In Azure DevOps, go to your team's project Summary page at <https://<server name>/<organization name>/<team name>>, for example, <https://dev.azure.com/DataScienceUnit/MyTeam>.
2. Select **Repos** in the left navigation, and at the top of the page, select the repository you want to clone.
3. On the repo page, select **Clone** at upper right.
4. In the **Clone repository** dialog, select **HTTPS** for an HTTP connection, or **SSH** for an SSH connection, and copy the clone URL under **Command line** to your clipboard.



5. On your local machine or DSVM, create the following directories:
 - For Windows: `C:\GitRepos`
 - For Linux: `$home/GitRepos`
6. Change to the directory you created.
7. In Git Bash, run the command `git clone <clone URL>` for each repository you want to clone.

For example, the following command clones the **TeamUtilities** repository to the *MyTeam* directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/TeamUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/MyTeam/TeamUtilities
```

8. Confirm that you can see the folders for the cloned repositories in your local project directory.

```
azureuser@LinuxDSVM1:~/Project_1$  
azureuser@LinuxDSVM1:~/Project_1$ dir  
GroupUtilities ProjectRepository TeamUtilities
```

Mount Azure file storage to your DSVM

If your team or project has shared assets in Azure file storage, mount the file storage to your local machine or DSVM. Follow the instructions at [Mount Azure file storage on your local machine or DSVM](#).

Next steps

Here are links to detailed descriptions of the other roles and tasks defined by the Team Data Science Process:

- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)

Team Data Science Process project planning

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process (TDSP) provides a lifecycle to structure the development of your data science projects. This article provides links to Microsoft Project and Excel templates that help you plan and manage these project stages.

The lifecycle outlines the major stages that projects typically execute, often iteratively:

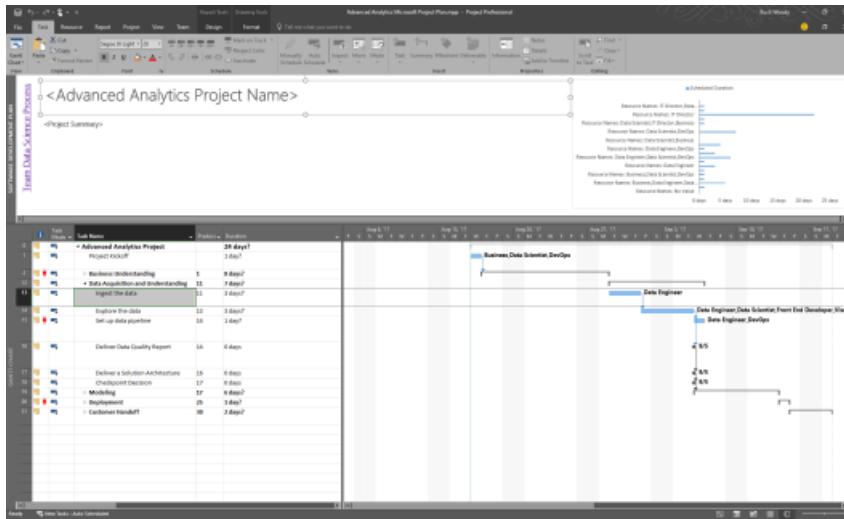
- Business Understanding
- Data Acquisition and Understanding
- Modeling
- Deployment
- Customer Acceptance

For descriptions of each of these stages, see [The Team Data Science Process lifecycle](#).

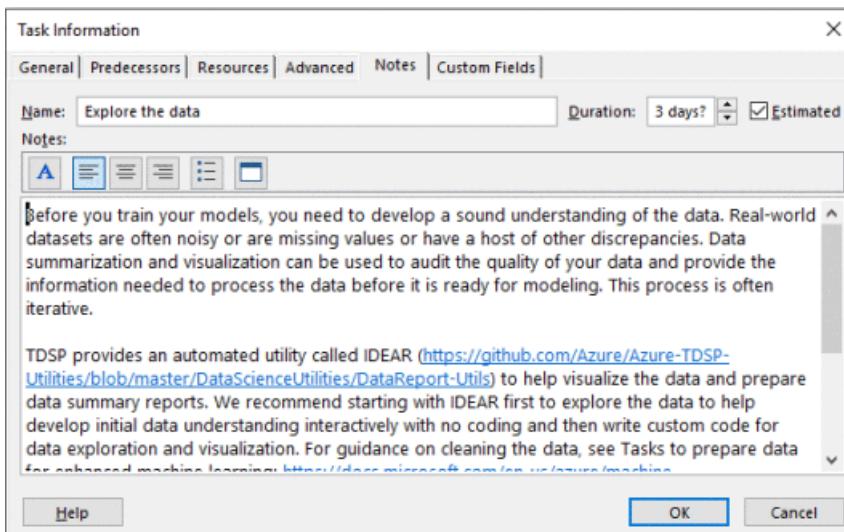
Microsoft Project template

The Microsoft Project template for the Team Data Science Process is available from here: [Microsoft Project template](#)

When you open the plan, click the link to the far left for the TDSP. Change the name and description and then add in any other team resources you need. Estimate the dates required from your experience.



Each task has a note. Open those tasks to see what resources have already been created for you.



Excel template

If don't have access to Microsoft Project, an Excel worksheet with all the same data is also available for download here: [Excel template](#) You can pull it in to whatever tool you prefer to use.

Use these templates at your own risk. The [usual disclaimers](#) apply.

Repository template

Use this [project template repository](#) to support efficient project execution and collaboration. This repository gives you a standardized directory structure and document templates you can use for your own TDSP project.

Next steps

[Agile development of data science projects](#) This document describes a data science project in a systematic, version controlled, and collaborative way by using the Team Data Science Process.

Related resources

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Agile development of data science projects

3/10/2022 • 7 minutes to read • [Edit Online](#)

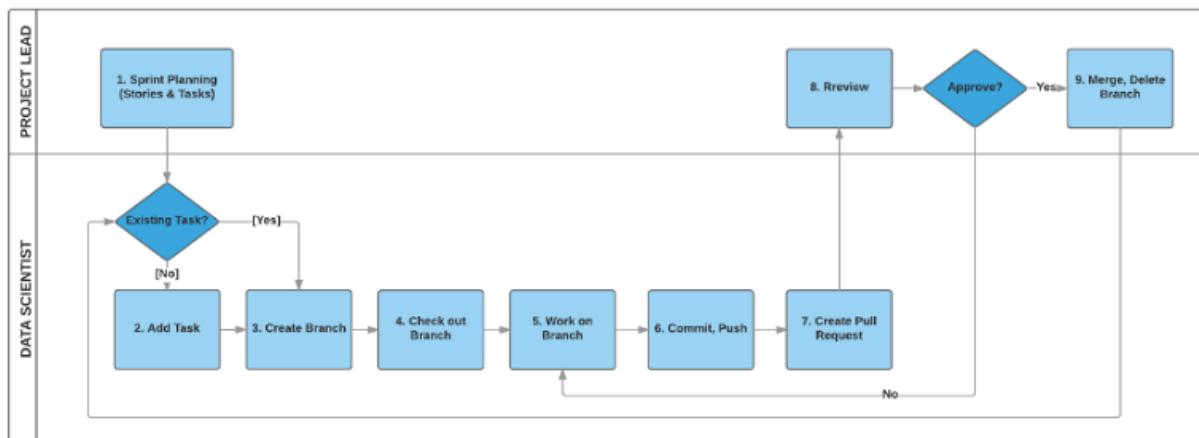
This document describes how developers can execute a data science project in a systematic, version controlled, and collaborative way within a project team by using the [Team Data Science Process](#) (TDSP). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities to efficiently execute cloud-based, predictive analytics solutions. For an outline of the roles and tasks that are handled by a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

This article includes instructions on how to:

- Do *sprint planning* for work items involved in a project.
- Add *work items* to sprints.
- Create and use an *agile-derived work item template* that specifically aligns with TDSP lifecycle stages.

The following instructions outline the steps needed to set up a TDSP team environment using Azure Boards and Azure Repos in Azure DevOps. The instructions use Azure DevOps because that is how to implement TDSP at Microsoft. If your group uses a different code hosting platform, the team lead tasks generally don't change, but the way to complete the tasks is different. For example, linking a work item with a Git branch might not be the same with GitHub as it is with Azure Repos.

The following figure illustrates a typical sprint planning, coding, and source-control workflow for a data science project:



Work item types

In the TDSP sprint planning framework, there are four frequently used *work item* types: *Features*, *User Stories*, *Tasks*, and *Bugs*. The backlog for all work items is at the project level, not the Git repository level.

Here are the definitions for the work item types:

- **Feature:** A Feature corresponds to a project engagement. Different engagements with a client are different Features, and it's best to consider different phases of a project as different Features. If you choose a schema such as `<ClientName>-<EngagementName>` to name your Features, you can easily recognize the context of the project and engagement from the names themselves.
- **User Story:** User Stories are work items needed to complete a Feature end-to-end. Examples of User Stories include:

- Get data
 - Explore data
 - Generate features
 - Build models
 - Operationalize models
 - Retrain models
- **Task:** Tasks are assignable work items that need to be done to complete a specific User Story. For example, Tasks in the User Story *Get data* could be:
 - Get SQL Server credentials
 - Upload data to Azure Synapse Analytics
 - **Bug:** Bugs are issues in existing code or documents that must be fixed to complete a Task. If Bugs are caused by missing work items, they can escalate to be User Stories or Tasks.

Data scientists may feel more comfortable using an agile template that replaces Features, User Stories, and Tasks with TDSP lifecycle stages and substages. To create an agile-derived template that specifically aligns with the TDSP lifecycle stages, see [Use an agile TDSP work template](#).

NOTE

TDSP borrows the concepts of Features, User Stories, Tasks, and Bugs from software code management (SCM). The TDSP concepts might differ slightly from their conventional SCM definitions.

Plan sprints

Many data scientists are engaged with multiple projects, which can take months to complete and proceed at different paces. Sprint planning is useful for project prioritization, and resource planning and allocation. In Azure Boards, you can easily create, manage, and track work items for your projects, and conduct sprint planning to ensure projects are moving forward as expected.

For more information about sprint planning, see [Scrum sprints](#).

For more information about sprint planning in Azure Boards, see [Assign backlog items to a sprint](#).

Add a Feature to the backlog

After your project and project code repository are created, you can add a Feature to the backlog to represent the work for your project.

1. From your project page, select **Boards > Backlogs** in the left navigation.
2. On the **Backlog** tab, if the work item type in the top bar is **Stories**, drop down and select **Features**. Then select **New Work Item**.

The screenshot shows the Azure DevOps interface for the 'FabrikamFiber Team Stories Backlog'. The left sidebar has a 'Backlogs' section highlighted with a red box. The main content area has a 'Backlog' tab selected. At the top right of the main area, there is a '+ New Work Item' button and a 'Features' tab, both highlighted with red boxes.

3. Enter a title for the Feature, usually your project name, and then select **Add to top**.

The screenshot shows the 'Backlog' page with a new feature named 'FabrikamFiber' added. The 'Add to top' button is highlighted with a red box.

4. From the Backlog list, select and open the new Feature. Fill in the description, assign a team member, and set planning parameters.

You can also link the Feature to the project's Azure Repos code repository by selecting **Add link** under the **Development** section.

After you edit the Feature, select **Save & Close**.

The screenshot shows the 'FabrikamFiber' feature details page. The 'Development' section is highlighted with a red box, containing a '+ Add link' button. Other sections like 'Description', 'Planning', and 'Related Work' are also visible.

Add a User Story to the Feature

Under the Feature, you can add User Stories to describe major steps needed to complete the project.

To add a new User Story to a Feature:

1. On the **Backlog** tab, select the + to the left of the Feature.

The screenshot shows the Azure DevOps Boards interface. At the top, it displays the organization path: fabrikam-org / FabrikamFiber / Boards / Backlogs. Below this, the team name 'FabrikamFiber Team' is shown with a dropdown arrow. The main navigation bar includes 'Backlog' (selected), 'Analytics', '+ New Work Item', and 'View as Board'. Under the backlog, there are columns for 'Order', 'Work Item Type', and 'Title'. A new item is being added, indicated by a red-bordered plus sign icon and the number '1'. The title of this new item is 'Feature' and the assignee is 'FabrikamFiber'. A tooltip 'Add: User Story' is visible near the bottom left of the backlog area.

2. Give the User Story a title, and edit details such as assignment, status, description, comments, planning, and priority.

You can also link the User Story to a branch of the project's Azure Repos code repository by selecting **Add link** under the **Development** section. Select the repository and branch you want to link the work item to, and then select **OK**.

The screenshot shows two overlapping windows. On the left is the 'Add link' dialog box, which has a title 'Add link' and a sub-instruction 'You are adding a link from: Get data'. It includes fields for 'Link type' (set to 'Branch'), 'Repository' (set to 'FabrikamFiber'), and 'Branch' (set to 'master'). There is also a 'Comment' field and 'OK' and 'Cancel' buttons. On the right is the 'Development' tab of a User Story details page. The tab header says 'Development' and has a 'Save & Close' button. Below the header, there is a 'Details' section showing 'Business' and 'Development' sections, with a 'Related Work' section below. A red box highlights the 'Development' tab header and the 'Add link' button in the 'Development' section of the details page.

3. When you're finished editing the User Story, select **Save & Close**.

Add a Task to a User Story

Tasks are specific detailed steps that are needed to complete each User Story. After all Tasks of a User Story are completed, the User Story should be completed too.

To add a Task to a User Story, select the + next to the User Story item, and select **Task**. Fill in the title and other information in the Task.

The screenshot shows the 'Backlog' view for the 'FabrikamFiber Team'. The backlog list contains one item: '1 Feature' under 'FabrikamFiber'. Below it, a dropdown menu is open, showing 'Task' highlighted with a red box. Other options in the menu include 'Get data' and 'Bug'.

After you create Features, User Stories, and Tasks, you can view them in the **Backlogs** or **Boards** views to track their status.

The screenshot shows the 'Backlogs' view for the 'FabrikamFiber Team'. The backlog list contains three items: '1 Feature' (State: New), 'User Story' (State: New), and 'Task' (State: New). The 'Backlogs' option is highlighted with a red box in the left navigation bar.

The screenshot shows the 'Boards' view for the 'FabrikamFiber Team'. The board interface displays work items in columns: 'New' (19 items), 'Active' (0/5), 'Resolved' (0/5), and 'Closed' (0/5). The 'Boards' option is highlighted with a red box in the left navigation bar.

Use an agile TDSP work template

Data scientists may feel more comfortable using an agile template that replaces Features, User Stories, and Tasks with TDSP lifecycle stages and substages. In Azure Boards, you can create an agile-derived template that uses TDSP lifecycle stages to create and track work items. The following steps walk through setting up a data science-specific agile process template and creating data science work items based on the template.

Set up an Agile Data Science Process template

1. From your Azure DevOps organization main page, select **Organization settings** from the left navigation.
2. In the **Organization Settings** left navigation, under **Boards**, select **Process**.

3. In the All processes pane, select the ... next to Agile, and then select Create inherited process.

The screenshot shows the 'Organization Settings' page in Azure DevOps. The left sidebar is collapsed, and the main area shows the 'All processes' list. The 'Agile (default)' item is selected, and its context menu is open, with the 'Create inherited process' option highlighted and surrounded by a red box. Other options in the menu include 'New team project' and 'Security'.

4. In the Create inherited process from Agile dialog, enter the name *AgileDataScienceProcess*, and select Create process.

The screenshot shows the 'Create inherited process from Agile' dialog. The 'Name' field contains 'AgileDataScienceProcess'. At the bottom, the 'Create process' button is highlighted with a red box.

5. In All processes, select the new AgileDataScienceProcess.

6. On the Work item types tab, disable Epic, Feature, User Story, and Task by selecting the ... next to each item and then selecting Disable.

All processes > AgileDataScienceProcess		Help	Filter by work item type name
Work item types	Backlog levels	Projects	
+ New work item type			
Name	Description		
Bug	Describes a divergence between required and actual behavior, and trac...		
Epic	Epics help teams effectively manage and groom their product backlog	Edit	Disable
Feature	I be released with the product		
Issue	Tracks difficult-to-progress.		
Task	Tracks work that needs to be done.		
Test Case	Server-side data for a set of steps to be tested.		
Test Plan	Tracks test activities for a specific milestone or release.		
Test Suite	Tracks test activites for a specific feature, requirement, or user story.		
User Story	Tracks an activity the user will be able to perform with the product		

7. In All processes, select the Backlog levels tab. Under Portfolios backlogs, select the ... next to Epic (disabled), and then select Edit/Rename.
8. In the Edit backlog level dialog box:
 - a. Under Name, replace Epic with TDSP Projects.
 - b. Under Work item types on this backlog level, select New work item type, enter TDSP Project, and select Add.
 - c. Under Default work item type, drop down and select TDSP Project.
 - d. Select Save.

Edit backlog level

The following fields are automatically added to all work item types on the Portfolio backlogs: Stack Rank

Name

Work item types on this backlog level

Epic (disabled)
 TDSP Project

[+ New work item type](#)

Default work item type

[Save](#) [Cancel](#)

9. Follow the same steps to rename Features to TDSP Stages, and add the following new work item types:

- *Business Understanding*
 - *Data Acquisition*
 - *Modeling*
 - *Deployment*
10. Under **Requirement backlog**, rename **Stories** to **TDSP Substages**, add the new work item type **TDSP Substage**, and set the default work item type to **TDSP Substage**.
11. Under **Iteration backlog**, add a new work item type **TDSP Task**, and set it to be the default work item type.

After you complete the steps, the backlog levels should look like this:

The screenshot shows the 'Backlog levels' section of the AgileDataScienceProcess. It includes three main sections: Portfolio backlogs, Requirement backlog, and Iteration backlog, each with a table mapping backlog levels to work item types.

Portfolio backlogs

Portfolio backlogs provide a way to group related items into a hierarchical structure. You can rename and edit any portfolio backlog level.

Backlog	Work item types
TDSP Projects	Epic (disabled)
TDSP Stages	TDSP Project (default) Business Understanding Data Acquisition Deployment Feature (disabled) Modeling

Requirement backlog

The requirement backlog level contains your base level work items. There is only one requirement backlog and it cannot be removed, but can be renamed and edited.

Backlog	Work item types
TDSP Substages	TDSP Substage (default) User Story (disabled)

Iteration backlog

The iteration backlog contains your task work items. There is only one level of iteration backlog and it cannot be removed. The iteration backlog does not have an associated color.

Backlog	Work item types
Tasks	Task (disabled) TDSP Task (default)

Create Agile Data Science Process work items

You can use the data science process template to create TDSP projects and track work items that correspond to TDSP lifecycle stages.

1. From your Azure DevOps organization main page, select **New project**.
2. In the **Create new project** dialog, give your project a name, and then select **Advanced**.
3. Under **Work item process**, drop down and select **AgileDataScienceProcess**, and then select **Create**.

Create new project

Project name *

 ✓

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Enterprise
[Members of your enterprise](#) can view the project.

Private
Only people you give access to will be able to view this project.

Advanced

Version control ?

Git

Work item process ?

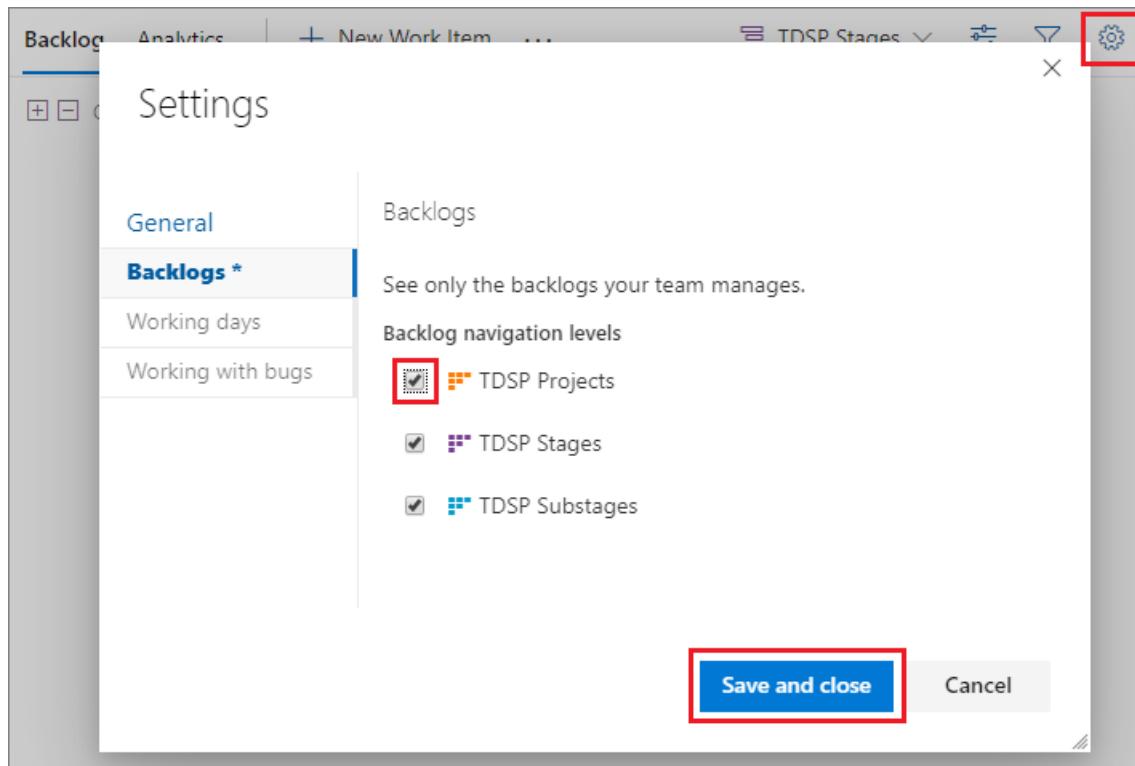
Agile

AgileDataScienceProcess

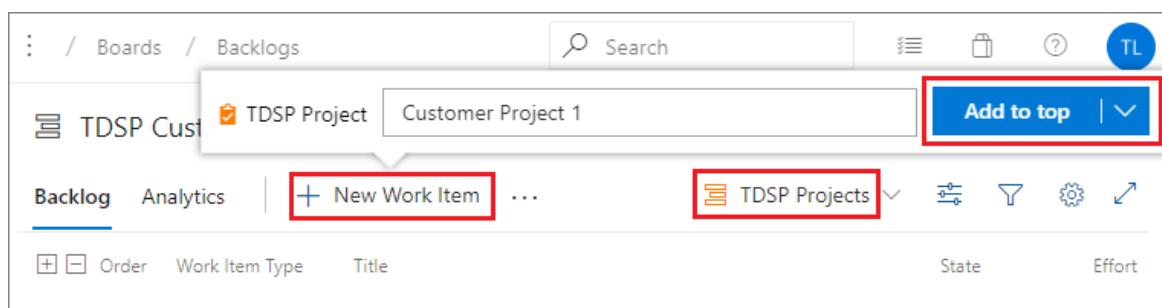
Basic
CMMI
Scrum

Cancel Create

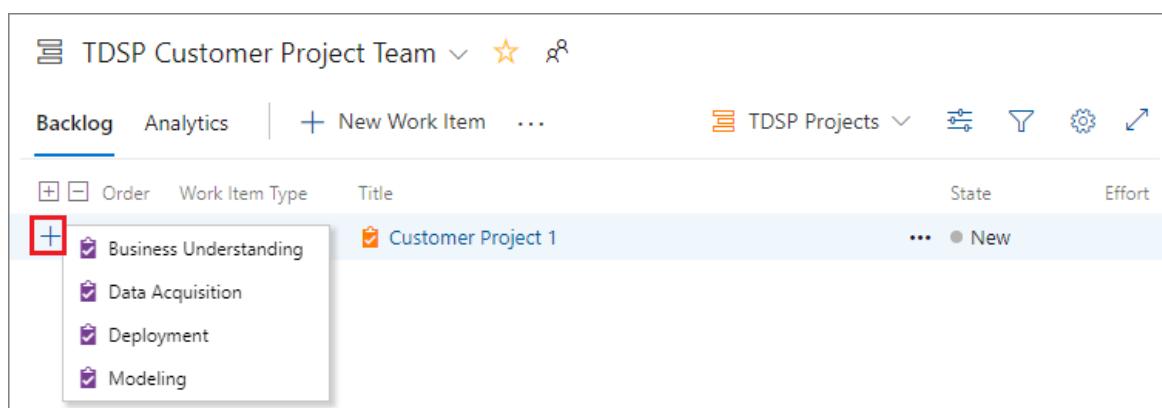
4. In the newly created project, select **Boards > Backlogs** in the left navigation.
5. To make TDSP Projects visible, select the **Configure team settings** icon. In the **Settings** screen, select the **TDSP Projects** check box, and then select **Save and close**.



6. To create a data science-specific TDSP Project, select TDSP Projects in the top bar, and then select New work item.
7. In the popup, give the TDSP Project work item a name, and select Add to top.



8. To add a work item under the TDSP Project, select the + next to the project, and then select the type of work item to create.



9. Fill in the details in the new work item, and select Save & Close.
10. Continue to select the + symbols next to work items to add new TDSP Stages, Substages, and Tasks.

Here is an example of how the data science project work items should appear in Backlogs view:

TDSP Customer Project Team star g8

Backlog Analytics | + New Work Item ... TDSP Projects Filter Settings

Order	Work Item Type	Title	State
1	TDSP Project	Customer Project 1	New
	Business Understanding	Define objectives	New
	TDSP Substage	Interview customers	New
	TDSP Task	Interview CEO	New
	TDSP Task	Interview CFO	New
	TDSP Substage	Prioritize tasks	New
	Data Acquisition	Get data	New
	TDSP Substage	Establish data connection	New
	TDSP Substage	Set up data store	New
	TDSP Task	Get datastore credentials	New

Next steps

- [Collaborative coding with Git](#) describes how to do collaborative code development for data science projects using Git as the shared code development framework, and how to link these coding activities to the work planned with the agile process.

Additional resources on agile processes:

- [Agile process](#)
- [Agile process work item types and workflow](#)

Related resources

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Collaborative coding with Git

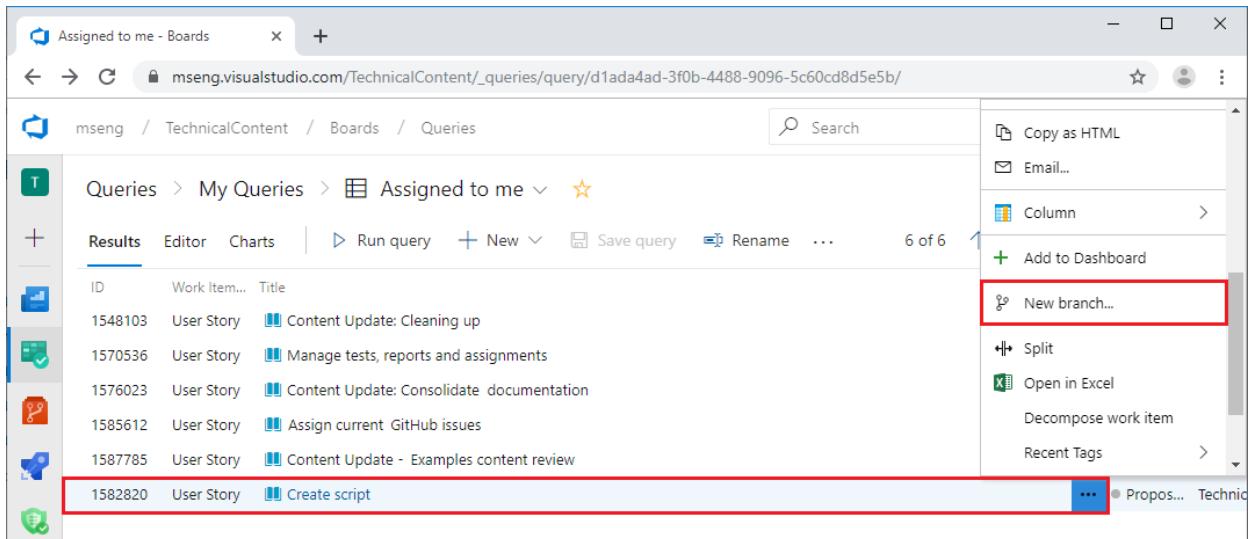
3/10/2022 • 4 minutes to read • [Edit Online](#)

This article describes how to use Git as the collaborative code development framework for data science projects. The article covers how to link code in Azure Repos to [agile development](#) work items in Azure Boards, how to do code reviews, and how to create and merge pull requests for changes.

Link a work item to an Azure Repos branch

Azure DevOps provides a convenient way to connect an Azure Boards User Story or Task work item with an Azure Repos Git repository branch. You can link your User Story or Task directly to the code associated with it.

To connect a work item to a new branch, select the **Actions** ellipsis (...) next to the work item, and on the context menu, scroll to and select **New branch...**.



In the **Create a branch** dialog, provide the new branch name and the base Azure Repos Git repository and branch. The base repository must be in the same Azure DevOps project as the work item. The base branch can be any existing branch. Select **Create branch**.

Create a branch

Name
script

Based on
TechnicalContent
master

Work items to link
Search work items by ID or title
1582820 Create script
Updated 10 minutes ago. • Proposed

Create branch Cancel

You can also create a new branch using the following Git bash command in Windows or Linux:

```
git checkout -b <new branch name> <base branch name>
```

If you don't specify a <base branch name>, the new branch is based on `main`.

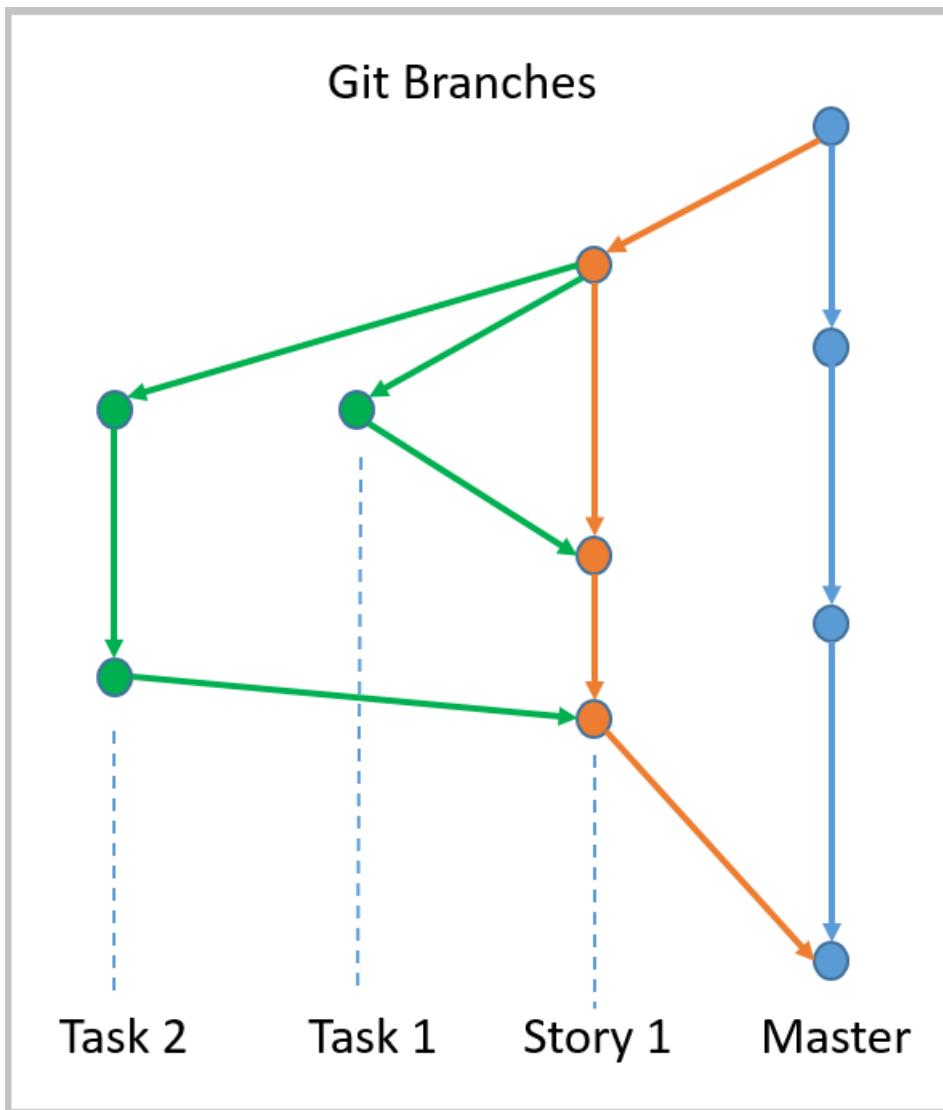
To switch to your working branch, run the following command:

```
git checkout <working branch name>
```

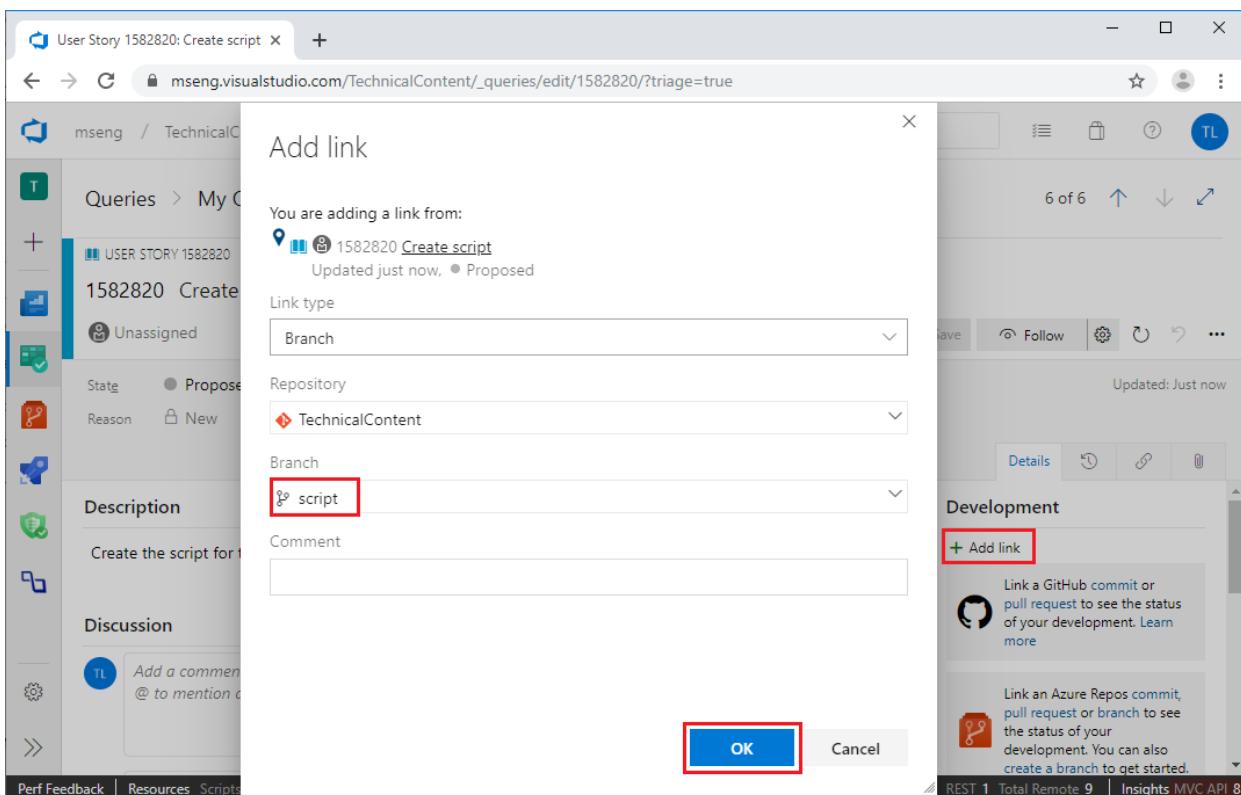
After you switch to the working branch, you can start developing code or documentation artifacts to complete the work item. Running `git checkout main` switches you back to the `main` branch.

It's a good practice to create a Git branch for each User Story work item. Then, for each Task work item, you can create a branch based on the User Story branch. Organize the branches in a hierarchy that corresponds to the User Story-Task relationship when you have multiple people working on different User Stories for the same project, or on different Tasks for the same User Story. You can minimize conflicts by having each team member work on a different branch, or on different code or other artifacts when sharing a branch.

The following diagram shows the recommended branching strategy for TDSP. You might not need as many branches as shown here, especially when only one or two people work on a project, or only one person works on all Tasks of a User Story. But separating the development branch from the primary branch is always a good practice, and can help prevent the release branch from being interrupted by development activities. For a complete description of the Git branch model, see [A Successful Git Branching Model](#).



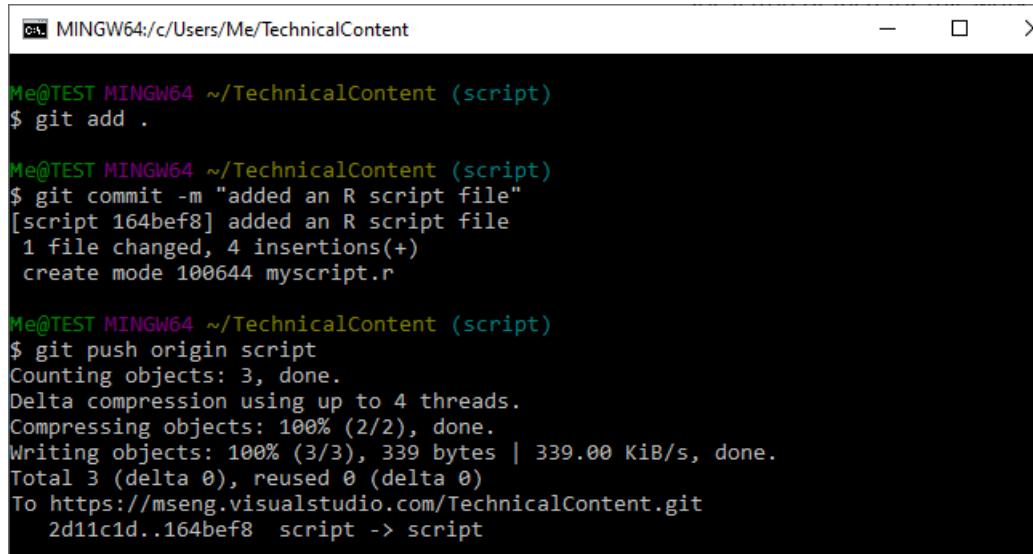
You can also link a work item to an existing branch. On the **Detail** page of a work item, select **Add link**. Then select an existing branch to link the work item to, and select **OK**.



Work on the branch and commit changes

After you make a change for your work item, such as adding an R script file to your local machine's `script` branch, you can commit the change from your local branch to the upstream working branch by using the following Git bash commands:

```
git status  
git add .  
git commit -m "added an R script file"  
git push origin script
```



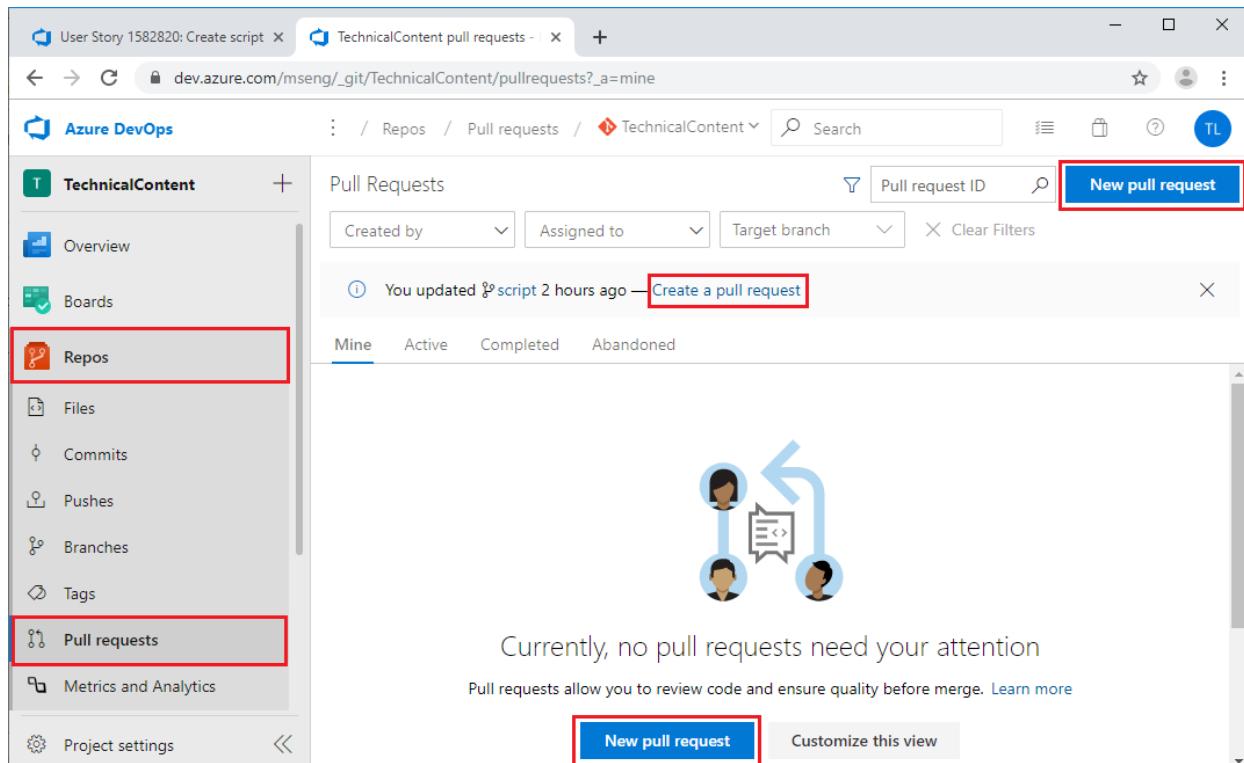
The screenshot shows a terminal window titled 'MINGW64:/c/Users/Me/TechnicalContent'. It displays the following command-line session:

```
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git add .  
  
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git commit -m "added an R script file"  
[script 164bef8] added an R script file  
1 file changed, 4 insertions(+)  
create mode 100644 myscript.r  
  
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git push origin script  
Counting objects: 3, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 339 bytes | 339.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To https://mseng.visualstudio.com/TechnicalContent.git  
 2d11c1d..164bef8  script -> script
```

Create a pull request

After one or more commits and pushes, when you're ready to merge your current working branch into its base branch, you can create and submit a *pull request* in Azure Repos.

From the main page of your Azure DevOps project, point to **Repos > Pull requests** in the left navigation. Then select either of the **New pull request** buttons, or the **Create a pull request** link.



The screenshot shows the 'Pull requests' page in the Azure DevOps interface. The left sidebar is highlighted with red boxes around the 'Repos' and 'Pull requests' sections. The main area shows a summary message: 'You updated script 2 hours ago — [Create a pull request](#)'. Below this, there are tabs for 'Mine', 'Active', 'Completed', and 'Abandoned'. A large 'New pull request' button is located at the bottom right of the page, also highlighted with a red box. The overall layout includes a header bar with tabs for 'User Story 1582820: Create script' and 'TechnicalContent pull requests', and a search bar.

On the New Pull Request screen, if necessary, navigate to the Git repository and branch you want to merge your changes into. Add or change any other information you want. Under **Reviewers**, add the names of the reviewers, and then select **Create**.

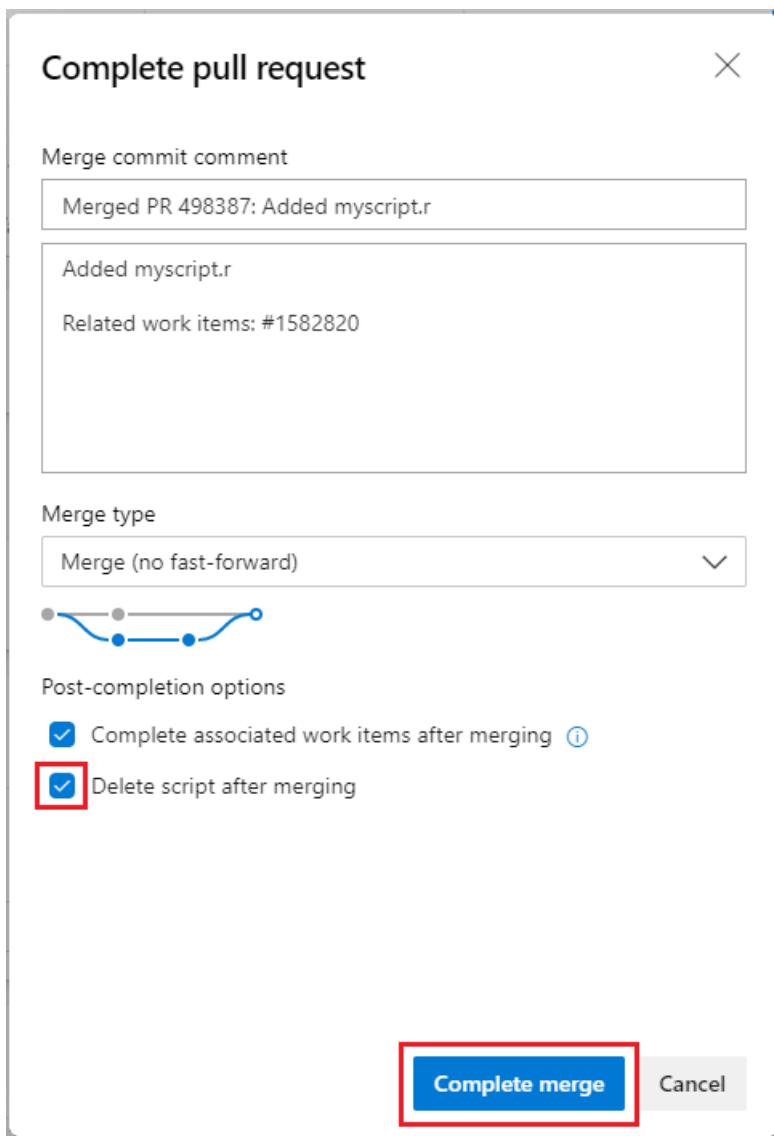
The screenshot shows the 'New Pull Request' interface in Azure DevOps. On the left, there's a sidebar with various project navigation links like Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests, Metrics and Analytics, Push Notifications, Pipelines, Compliance, and Project settings. The 'Pull requests' link is currently selected. The main area is titled 'New Pull Request' and shows a summary: 'script' into 'master'. Below this, there are fields for 'Title' (containing 'Added myscript.r') and 'Description' (containing 'Added myscript.r'). A note says 'Markdown supported.' There's also a rich text editor toolbar. The 'Reviewers' section contains two entries: 'Admin' and '[Documentation]\Test', with a placeholder 'Search users and groups to add as reviewers'. The 'Work Items' section has a search bar with '1582820 Create script' listed. At the bottom right is a large blue 'Create' button.

Review and merge

Once you create the pull request, your reviewers get an email notification to review the pull request. The reviewers test whether the changes work, and check the changes with the requester if possible. The reviewers can make comments, request changes, and approve or reject the pull request based on their assessment.

The screenshot shows the Azure DevOps interface for a pull request. The left sidebar is visible with options like Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests (which is selected), Metrics and Analytics, Push Notifications, Pipelines, Compliance, and Project settings. The main area displays a pull request titled "498387 ACTIVE Added myscript.r". The "Test" reviewer has commented: "I need to review this". The "Test" reviewer has also commented: "OK.". A third comment from "Tester" says: "Can you comment ?". On the right, there's a dropdown menu with options: Approve, Approve with suggestions, Wait for author, Reject, and Reset feedback. Below the dropdown is a "Create script" button. The "Labels" section shows an "Add label" button.

After the reviewers approve the changes, you or someone else with merge permissions can merge the working branch to its base branch. Select **Complete**, and then select **Complete merge** in the **Complete pull request** dialog. You can choose to delete the working branch after it has merged.



Confirm that the request is marked as COMPLETED.

498387 **COMPLETED** Added myscript.r

Test script into master

Delete source branch

Overview Files Updates Commits Additional Validations Conflicts

When you go back to **Repos** in the left navigation, you can see that you've been switched to the main branch since the `script` branch was deleted.

User Story 1582820: Create script X TechnicalContent - Repos X +

dev.azure.com/mseng/_git/TechnicalContent

Azure DevOps / Repos / Files / TechnicalContent

TechnicalContent + master TechnicalContent / Type to find a file or folder... Fork Clone

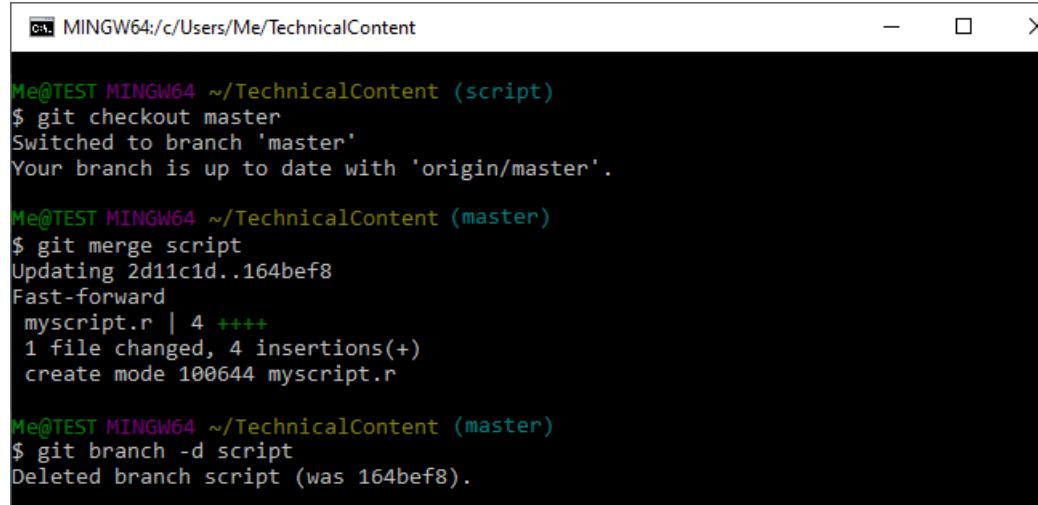
Overview Boards Repos Files Commits Pushes

Branch script was deleted so we've switched you to master, the default branch.

Name	Last change	Commits
myscript.r	3 hours ago	67a2baac
README.md	8/12/2019	f954d0c1
TeamList.md	8/16/2019	882421ed

You can also use the following Git bash commands to merge the `script` working branch to its base branch and delete the working branch after merging:

```
git checkout main
git merge script
git branch -d script
```



```
Me@TEST MINGW64 ~/TechnicalContent (script)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Me@TEST MINGW64 ~/TechnicalContent (master)
$ git merge script
Updating 2d11c1d..164bef8
Fast-forward
 myscript.r | 4 +++
 1 file changed, 4 insertions(+)
 create mode 100644 myscript.r

Me@TEST MINGW64 ~/TechnicalContent (master)
$ git branch -d script
Deleted branch script (was 164bef8).
```

Next steps

[Execute data science tasks](#) shows how to use utilities to complete several common data science tasks, such as interactive data exploration, data analysis, reporting, and model creation.

Related resources

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Execute data science tasks: exploration, modeling, and deployment

3/10/2022 • 2 minutes to read • [Edit Online](#)

Typical data science tasks include data exploration, modeling, and deployment. This article outlines the tasks to complete several common data science tasks such as interactive data exploration, data analysis, reporting, and model creation. Options for deploying a model into a production environment may include:

- Recommended: [Azure Machine Learning](#)
- Possible: [SQL-Server with ML services](#)

1. Exploration

A data scientist can perform exploration and reporting in a variety of ways: by using libraries and packages available for Python (matplotlib for example) or with R (ggplot or lattice for example). Data scientists can customize such code to fit the needs of data exploration for specific scenarios. The needs for dealing with structured data are different than for unstructured data such as text or images.

Products such as Azure Machine Learning also provide [advanced data preparation](#) for data wrangling and exploration, including feature creation. The user should decide on the tools, libraries, and packages that best suite their needs.

The deliverable at the end of this phase is a data exploration report. The report should provide a fairly comprehensive view of the data to be used for modeling and an assessment of whether the data is suitable to proceed to the modeling step.

2. Modeling

There are numerous toolkits and packages for training models in a variety of languages. Data scientists should feel free to use which ever ones they are comfortable with, as long as performance considerations regarding accuracy and latency are satisfied for the relevant business use cases and production scenarios.

Model management

After multiple models have been built, you usually need to have a system for registering and managing the models. Typically you need a combination of scripts or APIs and a backend database or versioning system. Azure Machine Learning provides [deployment of ONNX models](#) or [deployment of ML Flow models](#).

3. Deployment

Production deployment enables a model to play an active role in a business. Predictions from a deployed model can be used for business decisions.

Production platforms

There are various approaches and platforms to put models into production. We recommend [deployment to Azure Machine Learning](#).

NOTE

Prior to deployment, one has to insure the latency of model scoring is low enough to use in production.

A/B testing

When multiple models are in production, it can be useful to perform [A/B testing](#) to compare performance of the models.

Next steps

[Track progress of data science projects](#) shows how a data scientist can track the progress of a data science project.

[Model operation and CI/CD](#) shows how CI/CD can be performed with developed models.

Test data science code with Azure DevOps

3/10/2022 • 4 minutes to read • [Edit Online](#)

This article gives preliminary guidelines for testing code in a data science workflow, using Azure DevOps. Such testing gives data scientists a systematic and efficient way to check the quality and expected outcome of their code. We use a Team Data Science Process (TDSP) [project that uses the UCI Adult Income dataset](#) that we published earlier to show how code testing can be done.

Introduction on code testing

"Unit testing" is a longstanding practice for software development. But for data science, it's often not clear what "unit testing" means and how you should test code for different stages of a data science lifecycle, such as:

- Data preparation
- Data quality examination
- Modeling
- Model deployment

This article replaces the term "unit testing" with "code testing." It refers to testing as the functions that help to assess if code for a certain step of a data science lifecycle is producing results "as expected." The person who's writing the test defines what's "as expected," depending on the outcome of the function--for example, data quality check or modeling.

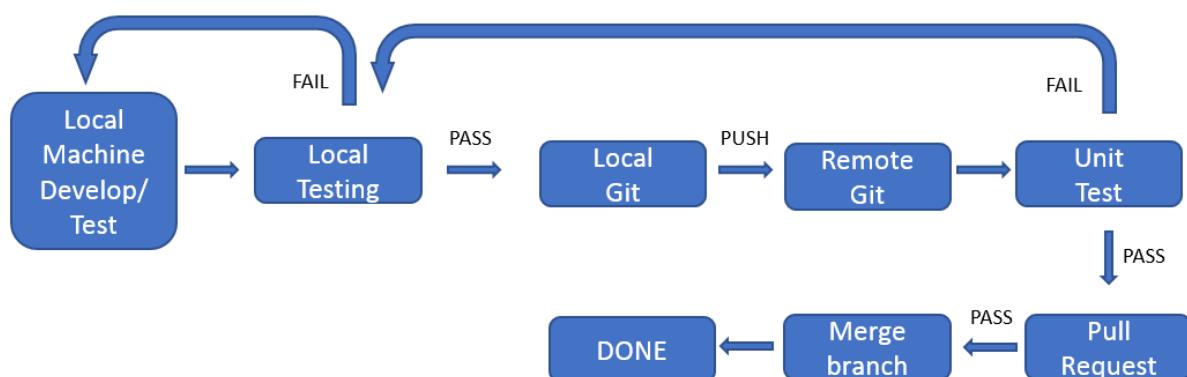
This article provides references as useful resources.

Azure DevOps for the testing framework

This article describes how to perform and automate testing by using Azure DevOps. You might decide to use alternative tools. We also show how to set up an automatic build by using Azure DevOps and build agents. For build agents, we use Azure Data Science Virtual Machines (DSVMs).

Flow of code testing

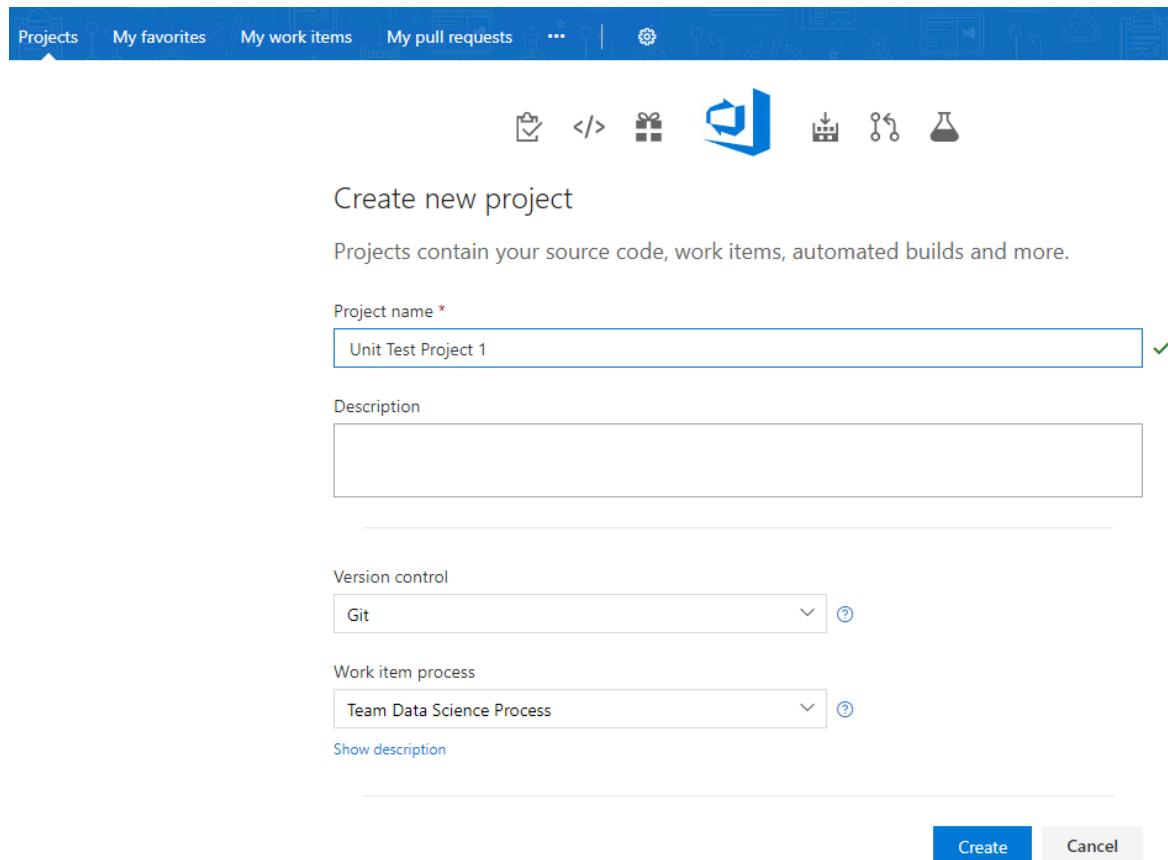
The overall workflow of testing code in a data science project looks like this:



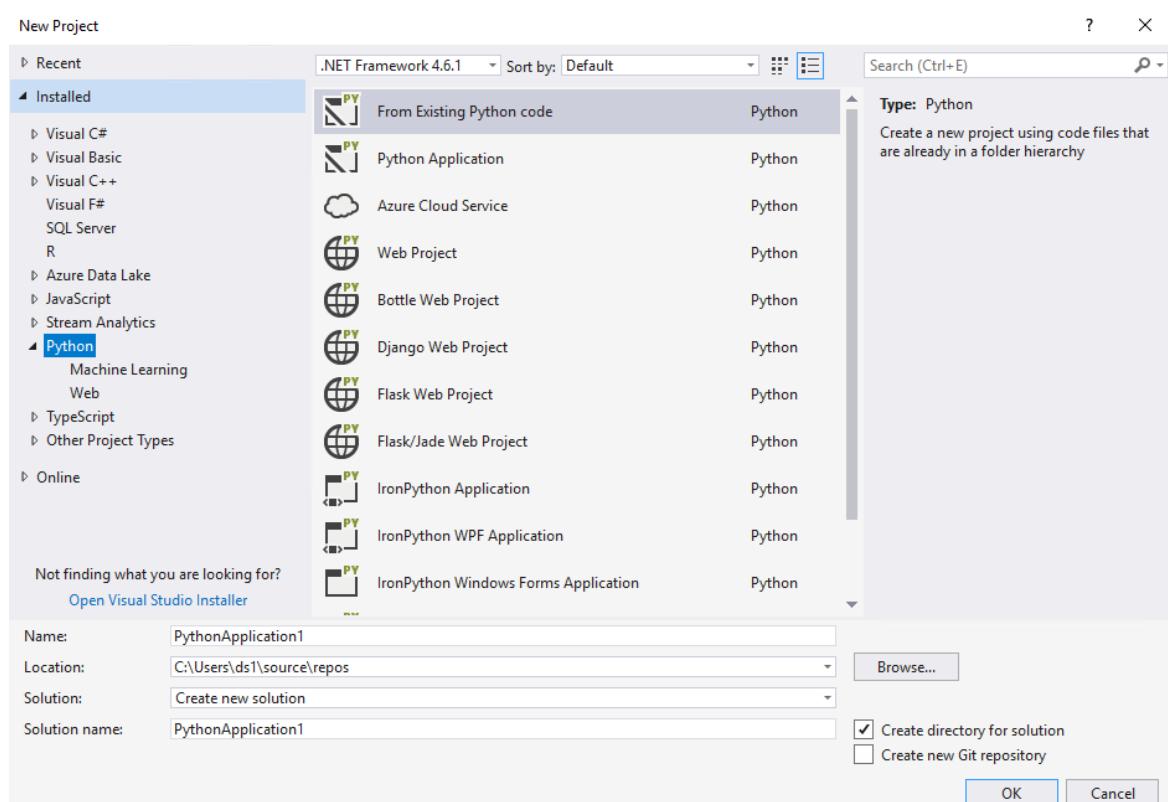
Detailed steps

Use the following steps to set up and run code testing and an automated build by using a build agent and Azure DevOps:

1. Create a project in the Visual Studio desktop application:



After you create your project, you'll find it in Solution Explorer in the right pane:



The screenshot shows the Microsoft Visual Studio Code interface. On the left, the code editor displays `test1.py` with several functions for data validation. On the right, the Solution Explorer panel shows a project named `unit_test_1` containing files like `data_ingestion.py`, `test_funcs.py`, and `test1.py`.

```

1 #class check_data(object):
2 #     """description of class"""
3 import pandas as pd
4 import pickle
5 import numpy as np
6
7
8 def checkColumnNames(csv_file, required_columns):
9     df = pd.read_csv(csv_file)
10    if set(df.columns) == set(required_columns):
11        print("Columns match!")
12    else:
13        print("Columns do not matched!")
14    return set(df.columns) == set(required_columns)
15
16 def checkResponseLevels(csv_file, response_column):
17     df = pd.read_csv(csv_file)
18     levels=list(df[response_column].unique())
19     if levels == [0,1]:
20         print("Levels match!")
21     else:
22         print("Levels do not match!")
23     return levels == [0,1]
24
25 def checkResponsePercent(csv_file, response_column):
26     df = pd.read_csv(csv_file)

```

2. Feed your project code into the Azure DevOps project code repository:

The screenshot shows the Azure DevOps GitHub interface for the repository `Unit_Test_UCI_Income`. It displays the file structure and a list of commits.

Name	Last change	Commits
<code>__pycache__</code>	3/14/2018	7785cf02 modified for py35 wei
<code>.vs</code>	3/14/2018	7785cf02 modified for py35 wei
<code>adult_income_model.pkl</code>	3/14/2018	ecf9822b add testing for model prediction wei
<code>adult_income_to_score.csv</code>	3/14/2018	0fcfa1fc7 add testing for model prediction wei
<code>data_ingestion.py</code>	1/11/2018	254c6c6f put some code wei
<code>test1.py</code>		

3. Suppose you've done some data preparation work, such as data ingestion, feature engineering, and creating label columns. You want to make sure your code is generating the results that you expect. Here's some code that you can use to test whether the data-processing code is working properly:

- Check that column names are right:

```

def checkColumnNames(csv_file, required_columns):
    df = pd.read_csv(csv_file)
    if set(df.columns) == set(required_columns):
        print("Columns match!")
    else:
        print("Columns do not matched!")
    return set(df.columns) == set(required_columns)

```

Check that response levels are right:

```

def checkResponseLevels(csv_file, response_column):
    df = pd.read_csv(csv_file)
    levels=list(df[response_column].unique())
    if levels == [0,1]:
        print("Levels match!")
    else:
        print("Levels do not match!")
    return levels == [0,1]

```

- Check that response percentage is reasonable:

```

def checkResponsePercent(csv_file,response_column):
    df = pd.read_csv(csv_file)
    ratio1 = df.loc[df[response_column] == 1].shape[0]/df.shape[0]
    if ratio1 > 0.5:
        print("Response levels(0/1) are messed up!")
    else:
        print("Response 0/1 are OK, and OK, Happy Friday!")
    return ratio1 < 0.5

```

- Check the missing rate of each column in the data:

```

def checkMissingRate2(csv_file, threshold):
    df = pd.read_csv(csv_file)
    for x in df.columns:
        miss_rate = df[x].isnull().sum()/df.shape[0]
        if miss_rate > threshold:
            print("{} has more than {} missing values!".format(x,threshold))
            return False
    print("No column has missing values more than {}!".format(threshold))
    return True

```

4. After you've done the data processing and feature engineering work, and you've trained a good model, make sure that the model you trained can score new datasets correctly. You can use the following two tests to check the prediction levels and distribution of label values:

- Check prediction levels:

```

def checkPredictionLevels(csv_file, model_file):
    df = pd.read_csv(csv_file)
    X_to_score = df[['education_num','age','hours_per_week']].values
    loaded_model = pickle.load(open(model_file, 'rb'))
    y_hat = loaded_model.predict(X_to_score)
    if list(set(y_hat)) == [0,1]:
        print("Prediction Levels match!")
    else:
        print("Prediction Levels do not match!")
    return list(set(y_hat)) == [0,1]

```

- Check the distribution of prediction values:

```

def checkPredictionPercent(csv_file,model_file):
    df = pd.read_csv(csv_file)
    X_to_score = df[['education_num','age','hours_per_week']].values
    loaded_model = pickle.load(open(model_file, 'rb'))
    y_hat = loaded_model.predict(X_to_score)
    ratio1 = sum(y_hat == 1)/len(y_hat)
    if ratio1 > 0.5:
        print("Response levels(0/1) are messed up!")
    else:
        print("Response 0/1 percent is OK, Happy prediction!")
    return ratio1 < 0.5

```

5. Put all test functions together into a Python script called `test_funcs.py`:

test_funcs.py

```
1 #class check_data(object):
2 #    """description of class"""
3 import pandas as pd
4 import pickle
5 import numpy as np
6
7
8 def checkColumnNames(csv_file, required_columns):
9     df = pd.read_csv(csv_file)
10    if set(df.columns) == set(required_columns):
11        print("Columns match!")
12    else:
13        print("Columns do not matched!")
14    return set(df.columns) == set(required_columns)
15
16 def checkResponseLevels(csv_file, response_column):
17    df = pd.read_csv(csv_file)
18    levels=list(df[response_column].unique())
19    if levels == [0,1]:
20        print("Levels match!")
21    else:
22        print("Levels do not match!")
23    return levels == [0,1]
24
25 def checkResponsePercent(csv_file, response_column):
26    df = pd.read_csv(csv_file)
27    ratio1 = df.loc[df[response_column] == 1].shape[0]/df.shape[0]
28    if ratio1 > 0.5:
29        print("Response levels(0/1) are messed up!")
30    else:
31        print("Response 0/1 are OK, and OK, Happy Friday!")
32    return ratio1 < 0.5
33
```

6. After the test codes are prepared, you can set up the testing environment in Visual Studio.

Create a Python file called **test1.py**. In this file, create a class that includes all the tests you want to do. The following example shows six tests prepared:

```
6 class Test_A(unittest.TestCase):
7
8     def checkColumnNames(self):
9         assert checkColumnNames(file_name, required_columns) == True
10
11    def checkResponseLevels(self):
12        assert checkResponseLevels(file_name, response_column) == True
13
14    def checkResponsePercent(self):
15        assert checkResponsePercent(file_name, response_column) == True
16
17    def checkMissingRate(self):
18        assert checkMissingRate2(file_name, threshold) == True
19
20    def checkPredictionLevels(self):
21        assert checkPredictionLevels(score_file_name, model_file_name) == True
22
23    def checkPredictionPercent(self):
24        assert checkPredictionPercent(score_file_name, model_file_name) == True
~~
```

1. Those tests can be automatically discovered if you put `codetest.testCase` after your class name. Open Test Explorer in the right pane, and select Run All. All the tests will run sequentially and will tell you if the test is successful or not.

The screenshot shows the Visual Studio Test Explorer interface. On the left, a tree view lists test cases categorized by execution time: 'Fast < 100 ms (1)' containing 'test_checkPredictionLevels' and 'Not Run (5)' containing 'test_checkColumnNames', 'test_checkMissingRate', 'test_checkPredictionPercent', 'test_checkResponseLevels', and 'test_checkResponsePercent'. On the right, the code editor displays the Python file 'test1.py' with several test functions defined:

```

1 import unittest
2 import pandas as pd
3 from test_funcs import *
4
5 class UCI_unit_test(unittest.TestCase):
6     def test_checkColumnNames(self):
7         assert checkColumnNames()
8
9     def test_checkResponseLevel(self):
10        assert checkResponseLevel()
11
12     def test_checkResponsePercent(self):
13        assert checkResponsePercent()
14
15     def test_checkMissingRate(self):
16        assert checkMissingRate()
17
18     def test_checkPredictionLevel(self):
19        assert checkPredictionLevel()
20
21     def test_checkPredictionPercent(self):
22        assert checkPredictionPercent()
23

```

- Check in your code to the project repository by using Git commands. Your most recent work will be reflected shortly in Azure DevOps.

```

PS C:\Unit_Test_UCI_Income> git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   .vs/unit_test_1/v15/.suo
    modified:   __pycache__/_test_funcs.cpython-35.pyc
    modified:   test1.py
    modified:   test1.pyc
    modified:   test_funcs.py
    modified:   unit_test.pyproj

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    __pycache__/_test1.cpython-35.pyc
    __pycache__/_test2.cpython-35.pyc
    __pycache__/_test_model.cpython-35.pyc

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Unit_Test_UCI_Income> git add .
PS C:\Unit_Test_UCI_Income> git commit -m"revised code we"
[master 60841e1] revised code we
 9 files changed, 7 insertions(+), 19 deletions(-)
 rewrite .vs/unit_test_1/v15/.suo (63%)
 create mode 100644 __pycache__/_test1.cpython-35.pyc
 create mode 100644 __pycache__/_test2.cpython-35.pyc
 create mode 100644 __pycache__/_test_model.cpython-35.pyc
 rewrite test1.pyc (64%)
PS C:\Unit_Test_UCI_Income> git push
Counting objects: 15, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (15/15), 6.16 KiB | 2.05 MiB/s, done.
Total 15 (delta 6), reused 0 (delta 0)
remote: Analyzing objects... (15/15) (26 ms)
remote: Storing packfile... done (213 ms)
remote: Storing index... done (32 ms)
To https://dg-ads.visualstudio.com/_git/Unit_Test_UCI_Income
 7785cf0..60841e1 master -> master
PS C:\Unit_Test_UCI_Income>

```

Unit_Test_UCI_Income / Type to find a file or folder...				
Contents		History		
Name		Last change ↓	Commits	
e_	.vs	2 minutes ago	60841e16	revised code wei wei
ome_model.pkl	__pycache__	2 minutes ago	60841e16	revised code wei wei
income_to_score.csv	unit_test.pyproj	2 minutes ago	60841e16	revised code wei wei
estion.py	test1.py	2 minutes ago	60841e16	revised code wei wei
	test1.pyc	2 minutes ago	60841e16	revised code wei wei
	test_funcs.py	2 minutes ago	60841e16	revised code wei wei

3. Set up automatic build and test in Azure DevOps:

- a. In the project repository, select **Build and Release**, and then select **+ New** to create a new build process.

The screenshot shows the Azure DevOps interface with the 'Build and Release' tab selected. Below it, the 'Definitions' tab is active. At the top right, there are buttons for '+ New', '+ Import', and 'Security'. Below the tabs, there are buttons for 'Definitions' and 'Queued'.

- b. Follow the prompts to select your source code location, project name, repository, and branch information.

The screenshot shows the 'Select a source' step in the build setup wizard. It includes sections for 'Team project' (set to 'Unit_Test_UCI_Income'), 'Repository' (set to 'Unit_Test_UCI_Income'), and 'Default branch for manual and scheduled builds' (set to 'master'). A 'Continue' button is at the bottom.

- c. Select a template. Because there's no Python project template, start by selecting **Empty process**.

Select a template

Or start with an  [Empty process](#)



Azure Service Fabric Application with Docker Support

Build and package an Azure Service Fabric application that contains Docker images to be pushed to a Docker registry.



Azure Web App for Java

Build your Java WAR file and deploy it directly to an Azure Web App.



Container

Build an image and push to Docker or Azure Container registry.

d. Name the build and select the agent. You can choose the default here if you want to use a DSVM to complete the build process. For more information about setting agents, see [Build and release agents](#).

Agent phase 

 Remove

Display name  *

Phase 1

Agent selection 

Agent queue  | [Manage](#) 

<inherit from definition>



Demands 

Name	Condition	Value
DotNetFramework	exists	

 Add

Execution plan 

Parallelism 

None Multi-configuration Multi-agent

Timeout  

0

Additional options 

Allow scripts to access OAuth token 

Run this phase 

Only when all previous phases have succeeded



e. Select  in the left pane, to add a task for this build phase. Because we're going to run the Python script **test1.py** to complete all the checks, this task is using a PowerShell command to run Python code.

Tasks Variables Triggers Options Retention History

Process Build process

Get sources Unit_Test_UCI_Income master

Phase 1 Run on agent

Add tasks Refresh ×

PowerShell Run a PowerShell script by Microsoft Corporation Add Learn more

Service Fabric PowerShell Run a PowerShell script within the context of an Azure Service Fabric cluster connection.

f. In the PowerShell details, fill in the required information, such as the name and version of PowerShell. Choose **Inline Script** as the type.

In the box under **Inline Script**, you can type `python test1.py`. Make sure the environment variable is set up correctly for Python. If you need a different version or kernel of Python, you can explicitly specify the path as shown in the figure:

Tasks Variables Triggers Options Retention History

Process Build process

Get sources Unit_Test_UCI_Income master

Phase 1 Run on agent

PowerShell Script PowerShell Version 1.* Display name * PowerShell Script Type * Inline Script Arguments * Arguments Inline Script * # You can write your powershell scripts inline here. # You can also pass predefined and custom variables to this scripts using arguments Write-Host "Hello World" #python test1.py C:\Anaconda\envs\py35\python.exe ./test1.py

g. Select **Save & queue** to complete the build pipeline process.

ment Groups*



Save & queue ...

History

PowerShell ⓘ

Link settings

Remove

Version 1.*

Display name *

PowerShell Script

Type * ⓘ

Inline Script

Arguments ⓘ

Inline Script * ⓘ

Now every time a new commit is pushed to the code repository, the build process will start automatically. You can define any branch. The process runs the `test1.py` file in the agent machine to make sure that everything defined in the code runs correctly.

If alerts are set up correctly, you'll be notified in email when the build is finished. You can also check the build status in Azure DevOps. If it fails, you can check the details of the build and find out which piece is broken.

Reply Reply All Forward IM

Wed 3/21/2018 10:51 AM



Visual Studio Team Services

Unit_Test_UCI_Income Build 47 succeeded

To:

47 - Succeeded

[Open Build Report in Web Access](#)

Continuous Integration Build of Unit_Test_UCI_Income-CI (2) (Unit_Test_UCI_Income)
Ran for 0.3 minutes (Default), completed at Wed 03/21/2018 05:50 PM

Request Summary

Request 47

Completed

Summary

| Finalize build

0 error(s), 0 warning(s)

| Phase 1

0 error(s), 0 warning(s)

Notes:

- All dates and times are shown in UTC

We sent you this notification due to a default subscription | [Unsubscribe](#) | [View](#)

Provided by [Microsoft Visual Studio® Team Foundation Server](#)

Builds Releases Library Task Groups Deployment Groups*

✓ Build 47

✓ Phase 1

- ✓ Job
 - ✓ Initialize Job
 - ✓ Get Sources
 - ✓ PowerShell Script
 - ✓ Post Job Cleanup
- ✓ Finalize build
- ✓ Report build status

Unit_Test_UCI_Income-CI (2) / Build 47

Edit build definition Queue new build... ▾

Build succeeded

Build 47

Ran for 16 seconds (Default), com|

Summary Timeline Code coverage* Tests

Build details

Definition	Unit_Test_UCI_Income-CI (2) (edit)
Source	master
Source version	Commit 60841e16
Requested by	Microsoft.VisualStudio.Services.TFS on
Queue name	Default
Queued	Wednesday, March 21, 2018 5:50 PM
Started	Wednesday, March 21, 2018 5:50 PM
Finished	Wednesday, March 21, 2018 5:50 PM
Retained state	Build not retained

Next steps

- See the [UCI income prediction repository](#) for concrete examples of unit tests for data science scenarios.
- Follow the preceding outline and examples from the UCI income prediction scenario in your own data science projects.

References

- [Team Data Science Process](#)
- [Visual Studio Testing Tools](#)
- [Azure DevOps Testing Resources](#)
- [Data Science Virtual Machines](#)

Track the progress of data science projects

3/10/2022 • 2 minutes to read • [Edit Online](#)

Data science group managers, team leads, and project leads can track the progress of their projects. Managers want to know what work has been done, who did the work, and what work remains. Managing expectations is an important element of success.

Azure DevOps dashboards

If you're using Azure DevOps, you can build dashboards to track the activities and work items associated with a given Agile project. For more information about dashboards, see [Dashboards, reports, and widgets](#).

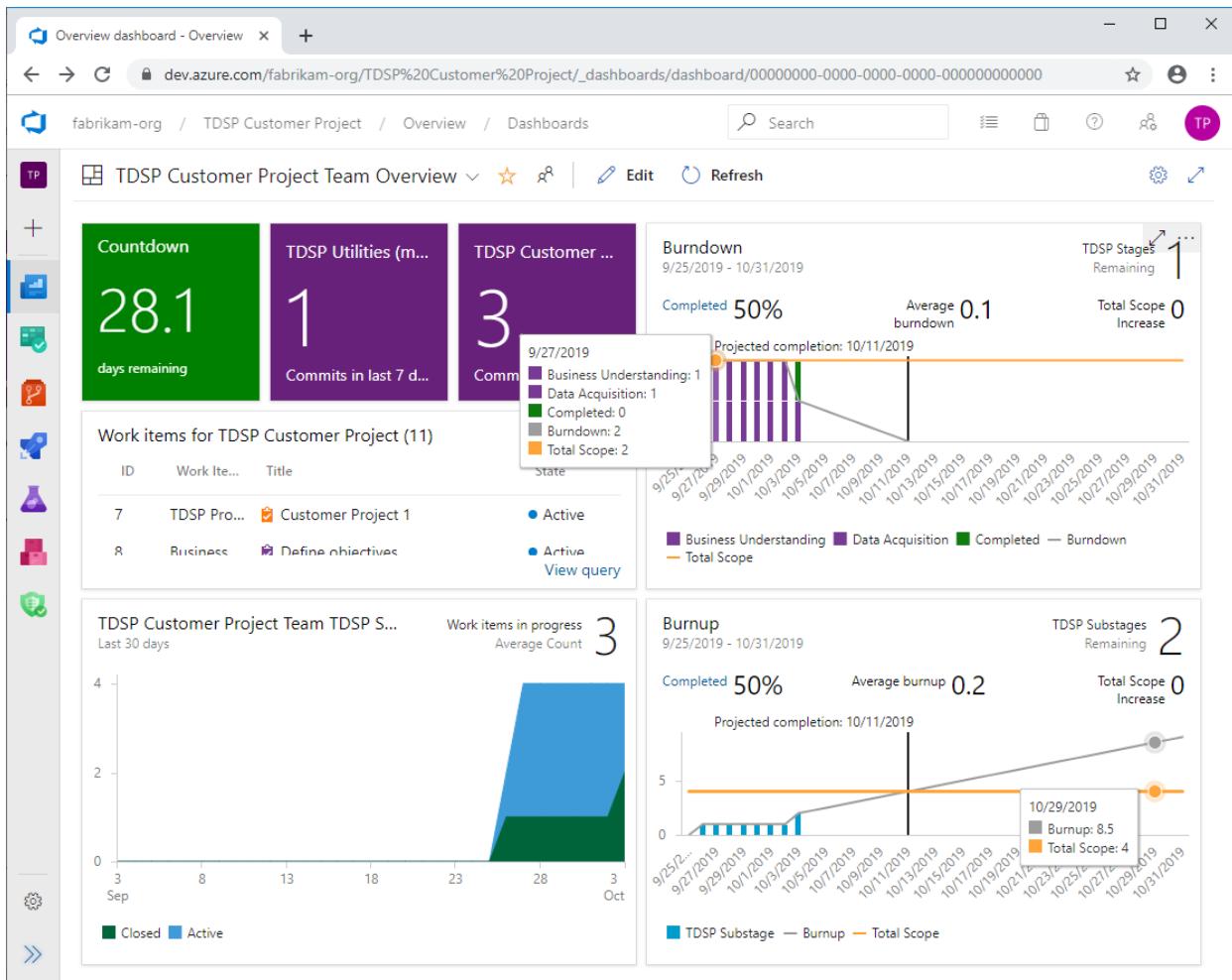
For instructions on how to create and customize dashboards and widgets in Azure DevOps, see the following quickstarts:

- [Add and manage dashboards](#)
- [Add widgets to a dashboard](#)

Example dashboard

Here is a simple example dashboard that tracks the sprint activities of an Agile data science project, including the number of commits to associated repositories.

- The **countdown** tile shows the number of days that remain in the current sprint.
- The two **code tiles** show the number of commits in the two project repositories for the past seven days.
- **Work items for TDSP Customer Project** shows the results of a query for all work items and their status.
- A **cumulative flow diagram** (CFD) shows the number of Closed and Active work items.
- The **burndown chart** shows work still to complete against remaining time in the sprint.
- The **burnup chart** shows completed work compared to total amount of work in the sprint.



Next steps

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Team Data Science Process for data scientists

3/10/2022 • 8 minutes to read • [Edit Online](#)

This article provides guidance to a set of objectives that are typically used to implement comprehensive data science solutions with Azure technologies. You are guided through:

- understanding an analytics workload
- using the Team Data Science Process
- using Azure Machine Learning
- the foundations of data transfer and storage
- providing data source documentation
- using tools for analytics processing

These training materials are related to the Team Data Science Process (TDSP) and Microsoft and open-source software and toolkits, which are helpful for envisioning, executing and delivering data science solutions.

Lesson Path

You can use the items in the following table to guide your own self-study. Read the *Description* column to follow the path, click on the *Topic* links for study references, and check your skills using the *Knowledge Check* column.

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Understand the processes for developing analytic projects	An introduction to the Team Data Science Process	We begin by covering an overview of the Team Data Science Process – the TDSP. This process guides you through each step of an analytics project. Read through each of these sections to learn more about the process and how you can implement it.	Review and download the TDSP Project Structure artifacts to your local machine for your project.
	Agile Development	The Team Data Science Process works well with many different programming methodologies. In this Learning Path, we use Agile software development. Read through the "What is Agile Development?" and "Building Agile Culture" articles, which cover the basics of working with Agile. There are also other references at this site where you can learn more.	Explain Continuous Integration and Continuous Delivery to a colleague.

Objective	Topic	Description	Knowledge Check
	DevOps for Data Science	<p>Developer Operations (DevOps) involves people, processes, and platforms you can use to work through a project and integrate your solution into an organization's standard IT. This integration is essential for adoption, safety, and security. In this online course, you learn about DevOps practices as well as understand some of the toolchain options you have.</p>	<p>Prepare a 30-minute presentation to a technical audience on how DevOps is essential for analytics projects.</p>
Understand the Technologies for Data Storage and Processing	Microsoft Business Analytics and AI	<p>We focus on a few technologies in this Learning Path that you can use to create an analytics solution, but Microsoft has many more. To understand the options you have, it's important to review the platforms and features available in Microsoft Azure, the Azure Stack, and on-premises options. Review this resource to learn the various tools you have available to answer analytics question.</p>	<p>Download and review the presentation materials from this workshop.</p>
Setup and Configure your training, development, and production environments	Microsoft Azure	<p>Now let's create an account in Microsoft Azure for training and learn how to create development and test environments. These free training resources get you started. Complete the "Beginner" and "Intermediate" paths.</p>	<p>If you do not have an Azure Account, create one. Log in to the Microsoft Azure portal and create one Resource Group for training.</p>
	The Microsoft Azure Command-Line Interface (CLI)	<p>There are multiple ways of working with Microsoft Azure – from graphical tools like VSCode and Visual Studio, to Web interfaces such as the Azure portal, and from the command line, such as Azure PowerShell commands and functions. In this article, we cover the Command-Line Interface (CLI), which you can use locally on your workstation, in Windows and other Operating Systems, as well as in the Azure portal.</p>	<p>Set your default subscription with the Azure CLI.</p>

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
	Microsoft Azure Storage	<p>You need a place to store your data. In this article, you learn about Microsoft Azure's storage options, how to create a storage account, and how to copy or move data to the cloud. Read through this introduction to learn more.</p>	<p>Create a Storage Account in your training Resource Group, create a container for a Blob object, and upload and download data.</p>
	Microsoft Azure Active Directory	<p>Microsoft Azure Active Directory (AAD) forms the basis of securing your application. In this article, you learn more about accounts, rights, and permissions. Active Directory and security are complex topics, so just read through this resource to understand the fundamentals.</p>	<p>Add one user to Azure Active Directory. NOTE: You may not have permissions for this action if you are not the administrator for the subscription. If that's the case, simply review this tutorial to learn more.</p>
	The Microsoft Azure Data Science Virtual Machine	<p>You can install the tools for working with Data Science locally on multiple operating systems. But the Microsoft Azure Data Science Virtual Machine (DSVM) contains all of the tools you need and plenty of project samples to work with. In this article, you learn more about the DVSM and how to work through its examples. This resource explains the Data Science Virtual Machine, how you can create one, and a few options for developing code with it. It also contains all the software you need to complete this learning path – so make sure you complete the Knowledge Path for this topic.</p>	<p>Create a Data Science Virtual Machine and work through at least one lab.</p>

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Install and Understand the tools and technologies for working with Data Science solutions	Working with git	To follow our DevOps process with the TDSP, we need to have a version-control system. Microsoft Azure Machine Learning uses git, a popular open-source distributed repository system. In this article, you learn more about how to install, configure, and work with git and a central repository – GitHub.	Clone this GitHub project for your learning path project structure.
	VSCode	VSCode is a cross-platform Integrated Development Environment (IDE) that you can use with multiple languages and Azure tools. You can use this single environment to create your entire solution. Watch these introductory videos to get started.	Install VSCode, and work through the VS Code features in the Interactive Editor Playground.
	Programming with Python	In this solution we use Python, one of the most popular languages in Data Science. This article covers the basics of writing analytic code with Python, and resources to learn more. Work through sections 1-9 of this reference, then check your knowledge.	Add one entity to an Azure Table using Python.
	Working with Notebooks	Notebooks are a way of introducing text and code in the same document. Azure Machine Learning work with Notebooks, so it is beneficial to understand how to use them. Read through this tutorial and give it a try in the Knowledge Check section.	Open this page, and click on the "Welcome to Python.ipynb" link. Work through the examples on that page.
	Machine Learning	Creating advanced Analytic solutions involves working with data, using Machine Learning, which also forms the basis of working with Artificial Intelligence and Deep Learning. This course teaches you more about Machine Learning. For a comprehensive course on Data Science, check out this certification.	Locate a resource on Machine Learning Algorithms. (Hint: Search on "azure machine learning algorithm cheat sheet")

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
	scikit-learn	The scikit-learn set of tools allows you to perform data science tasks in Python. We use this framework in our solution. This article covers the basics and explains where you can learn more.	Using the Iris dataset, persist an SVM model using Pickle.
	Working with Docker	Docker is a distributed platform used to build, ship, and run applications, and is used frequently in Azure Machine Learning. This article covers the basics of this technology and explains where you can go to learn more.	Open Visual Studio Code, and install the Docker Extension . Create a simple Node Docker container .
	HDInsight	HDInsight is the Hadoop open-source infrastructure, available as a service in Microsoft Azure. Your Machine Learning algorithms may involve large sets of data, and HDInsight has the ability to store, transfer and process data at large scale. This article covers working with HDInsight.	Create a small HDInsight cluster . Use HiveQL statements to project columns onto an /example/data/sample.log file . Alternatively, you can complete this knowledge check on your local system .
Create a Data Processing Flow from Business Requirements	Determining the Question, following the TDSP	With the development environment installed and configured, and the understanding of the technologies and processes in place, it's time to put everything together using the TDSP to perform an analysis. We need to start by defining the question, selecting the data sources, and the rest of the steps in the Team Data Science Process. Keep in mind the DevOps process as we work through this process. In this article, you learn how to take the requirements from your organization and create a data flow map through your application to define your solution using the Team Data Science Process	Locate a resource on " The 5 data science questions " and describe one question your organization might have in these areas. Which algorithms should you focus on for that question?

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Use Azure Machine Learning to create a predictive solution	Azure Machine Learning	Microsoft Azure Machine Learning uses AI for data wrangling and feature engineering, manages experiments, and tracks model runs. All of this works in a single environment and most functions can run locally or in Azure. You can use the PyTorch, TensorFlow, and other frameworks to create your experiments. In this article, we focus on a complete example of this process, using everything you've learned so far.	
Use Power BI to visualize results	Power BI	Power BI is Microsoft's data visualization tool. It is available on multiple platforms from Web to mobile devices and desktop computers. In this article, you learn how to work with the output of the solution you've created by accessing the results from Azure storage and creating visualizations using Power BI.	Complete this tutorial on Power BI . Then connect Power BI to the Blob CSV created in an experiment run.
Monitor your Solution	Application Insights	There are multiple tools you can use to monitor your end solution. Azure Application Insights makes it easy to integrate built-in monitoring into your solution.	Set up Application Insights to monitor an Application .
	Azure Monitor logs	Another method to monitor your application is to integrate it into your DevOps process. The Azure Monitor logs system provides a rich set of features to help you watch your analytic solutions after you deploy them.	Complete this tutorial on using Azure Monitor logs.
Complete this Learning Path		Congratulations! You've completed this learning path.	

Next steps

[Team Data Science Process for Developer Operations](#) This article explores the Developer Operations (DevOps) functions that are specific to an Advanced Analytics and Cognitive Services solution implementation.

Team Data Science Process for Developer Operations

3/10/2022 • 9 minutes to read • [Edit Online](#)

This article explores the Developer Operations (DevOps) functions that are specific to an Advanced Analytics and Cognitive Services solution implementation. These training materials implement the Team Data Science Process (TDSP) and Microsoft and open-source software and toolkits, helpful for envisioning, executing and delivering data science solutions. It references topics that cover the DevOps Toolchain that is specific to Data Science and AI projects and solutions.

Lesson Path

The following table provides level-based guidance to help complete the DevOps objectives for implementing data science solutions on Azure.

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
Understand Advanced Analytics	The Team Data Science Process Lifecycle	This technical walkthrough describes the Team Data Science Process	Data Science	Intermediate	General technology background, familiarity with data solutions, Familiarity with IT projects and solution implementation
Understand the Microsoft Azure Platform for Advanced Analytics	Information Management	This reference gives and overview of Azure Data Factory to build pipelines for analytics data solutions	Microsoft Azure Data Factory	Experienced	General technology background, familiarity with data solutions, Familiarity with IT projects and solution implementation
		This reference covers an overview of the Azure Data Catalog which you can use to document and manage metadata on your data sources	Microsoft Azure Data Catalog	Intermediate	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This reference covers an overview of the Azure Event Hubs system and how you can use it to ingest data into your solution	Azure Event Hubs	Intermediate	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources, familiarity with the Internet of Things (IoT) terminology and use
	Big Data Stores	This reference covers an overview of using the Azure Synapse Analytics to store and process large amounts of data	Azure Synapse Analytics	Experienced	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources, familiarity with HDFS terminology and use
		This reference covers an overview of using Azure Data Lake to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics	Azure Data Lake Store	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources, familiarity with HDFS

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
	Machine learning and analytics	This reference covers an introduction to machine learning, predictive analytics, and Artificial Intelligence systems	Azure Machine Learning	Intermediate	General technology background, familiarity with data solutions, familiarity with Data Science terms, familiarity with Machine Learning and artificial intelligence terms
		This article provides an introduction to Azure HDInsight, a cloud distribution of the Hadoop technology stack. It also covers what a Hadoop cluster is and when you would use it	Azure HDInsight	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources
		This reference covers an overview of the Azure Data Lake Analytics job service	Azure Data Lake Analytics	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources
		This overview covers using Azure Stream Analytics as a fully-managed event-processing engine to perform real-time analytic computations on streaming data	Azure Stream Analytics	Intermediate	General technology background, familiarity with data solutions, familiarity with structured and unstructured data concepts
	Intelligence	This reference covers an overview of the available Cognitive Services (such as vision, text, and search) and how to get started using them	Cognitive Services	Experienced	General technology background, familiarity with data solutions, software development

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This reference covers and introduction to the Microsoft Bot Framework and how to get started using it	Bot Framework	Experienced	General technology background, familiarity with data solutions
	Visualization	This self-paced, online course covers the Power BI system, and how to create and publish reports	Microsoft Power BI	Beginner	General technology background, familiarity with data solutions
	Solutions	This resource page covers multiple applications you can review, test and implement to see a complete solution from start to finish	Microsoft Azure, Azure Machine Learning, Cognitive Services, Microsoft R, Azure Cognitive Search, Python, Azure Data Factory, Power BI, Azure Document DB, Application Insights, Azure SQL DB, Azure Synapse Analytics, Microsoft SQL Server, Azure Data Lake, Cognitive Services, Bot Framework, Azure Batch,	Intermediate	General technology background, familiarity with data solutions
Understand and Implement DevOps processes	What is DevOps?	This article explains the fundamentals of DevOps and helps explain how they map to DevOps practices	DevOps, Microsoft Azure Platform, Azure DevOps	Intermediate	Familiarity with Agile and other Development Frameworks, IT Operations Familiarity
Use the DevOps Toolchain for Data Science	Configure	This reference covers the basics of choosing the proper visualization in Visio to communicate your project design	Visio	Intermediate	General technology background, familiarity with data solutions

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This reference describes the Azure Resource Manager, terms, and serves as the primary root source for samples, getting started, and other references	Azure Resource Manager, Azure PowerShell, Azure CLI	Intermediate	General technology background, familiarity with data solutions
		This reference explains the Azure Data Science Virtual Machines for Linux and Windows	Data Science Virtual Machine	Experienced	Familiarity with Data Science Workloads, Linux
		This walkthrough explains configuring Azure cloud service roles with Visual Studio - pay close attention to the connection strings specifically for storage accounts	Visual Studio	Intermediate	Software Development
		This series teaches you how to use Microsoft Project to schedule time, resources and goals for an Advanced Analytics project	Microsoft Project	Intermediate	Understand Project Management Fundamentals
		This Microsoft Project template provides a time, resources and goals tracking for an Advanced Analytics project	Microsoft Project	Intermediate	Understand Project Management Fundamentals
		This Azure Data Catalog tutorial describes a system of registration and discovery for enterprise data assets	Azure Data Catalog	Beginner	Familiarity with Data Sources and Structures

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This Microsoft Virtual Academy course explains how to set up Dev/Test with Visual Studio Codespace and Microsoft Azure	Visual Studio Codespace	Experienced	Software Development, familiarity with Dev/Test environments
		This Management Pack download for Microsoft System Center contains a Guidelines Document to assist in working with Azure assets	System Center	Intermediate	Experience with System Center for IT Management
		This document is intended for developer and operations teams to understand the benefits of PowerShell Desired State Configuration	PowerShell DSC	Intermediate	Experience with PowerShell coding, enterprise architectures, scripting
	Code	This download also contains documentation on using Visual Studio Codespace Code for creating Data Science and AI applications	Visual Studio Codespace	Intermediate	Software Development
		This getting started site teaches you about DevOps and Visual Studio	Visual Studio	Beginner	Software Development
		You can write code directly from the Azure portal using the App Service Editor. Learn more at this resource about Continuous Integration with this tool	Azure portal	Highly Experienced	Data Science background - but read this anyway

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This resource explains how to get started with Azure Machine Learning	Azure Machine Learning	Intermediate	Software Development
		This reference contains a list and a study link to all of the development tools on the Data Science Virtual Machine in Azure	Data Science Virtual Machine	Experienced	Software Development, Data Science
		Read and understand each of the references in this Azure Security Trust Center for Security, Privacy, and Compliance - VERY important	Azure Security	Intermediate	System Architecture Experience, Security Development experience
	Build	This course teaches you about enabling DevOps Practices with Visual Studio Codespace Build	Visual Studio Codespace	Experienced	Software Development, Familiarity with an SDLC
		This reference explains compiling and building using Visual Studio	Visual Studio	Intermediate	Software Development, Familiarity with an SDLC
		This reference explains how to orchestrate processes such as software builds with Runbooks	System Center	Experienced	Experience with System Center Orchestrator
	Test	Use this reference to understand how to use Visual Studio Codespace for Test Case Management	Visual Studio Codespace	Experienced	Software Development, Familiarity with an SDLC

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		Use this previous reference for Runbooks to automate tests using System Center	System Center	Experienced	Experience with System Center Orchestrator
		As part of not only testing but development, you should build in Security. The Microsoft SDL Threat Modeling Tool can help in all phases. Learn more and download it here	Threat Monitoring Tool	Experienced	Familiarity with security concepts, software development
		This article explains how to use the Microsoft Attack Surface Analyzer to test your Advanced Analytics solution	Attack Surface Analyzer	Experienced	Familiarity with security concepts, software development
	Package	This reference explains the concepts of working with Packages in TFS and Visual Studio Codespace	Visual Studio Codespace	Experienced	Software development, familiarity with an SDLC
		Use this previous reference for Runbooks to automate packaging using System Center	System Center	Experienced	Experience with System Center Orchestrator
		This reference explains how to create a data pipeline for your solution, which you can save as a JSON template as a "package"	Azure Data Factory	Intermediate	General computing background, data project experience
		This topic describes the structure of an Azure Resource Manager template	Azure Resource Manager	Intermediate	Familiarity with the Microsoft Azure Platform

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		DSC is a management platform in PowerShell that enables you to manage your IT and development infrastructure with configuration as code, saved as a package. This reference is an overview for that topic	PowerShell Desired State Configuration	Intermediate	PowerShell coding, familiarity with enterprise architectures, scripting
	Release	This head-reference article contains concepts for build, test, and release for CI/CD environments	Visual Studio Codespace	Experienced	Software development, familiarity with CI/CD environments, familiarity with an SDLC
		Use this previous reference for Runbooks to automate release management using System Center	System Center	Experienced	Experience with System Center Orchestrator
		This article helps you determine the best option to deploy the files for your web app, mobile app backend, or API app to Azure App Service, and then guides you to appropriate resources with instructions specific to your preferred option	Microsoft Azure Deployment	Intermediate	Software development, experience with the Microsoft Azure platform
	Monitor	This reference explains Application Insights and how you can add it to your Advanced Analytics Solutions	Application Insights	Intermediate	Software Development, familiarity with the Microsoft Azure platform

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This topic explains basic concepts about Operations Manager for the administrator who manages the Operations Manager infrastructure and the operator who monitors and supports the Advanced Analytics Solution	System Center	Experienced	Familiarity with enterprise monitoring, System Center Operations Manager
		This blog entry explains how to use the Azure Data Factory to monitor and manage the Advanced Analytics pipeline	Azure Data Factory	Intermediate	Familiarity with Azure Data Factory
Understand how to use Open Source DevOps Tools with DevOps on Azure	Open Source DevOps Tools and Azure	This reference page contains two videos and a whitepaper on using Chef with Azure deployments	Chef	Experienced	Familiarity with the Azure Platform, Familiarity with DevOps
		This site has a toolchain selection path	DevOps, Microsoft Azure Platform, Azure DevOps, Open Source Software	Experienced	Used an SDLC, familiarity with Agile and other Development Frameworks, IT Operations Familiarity
		This tutorial automates the build and test phase of application development using a continuous integration and deployment CI/CD pipeline	Jenkins	Experienced	Familiarity with the Azure Platform, Familiarity with DevOps, Familiarity with Jenkins

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This contains an overview of working with Docker and Azure as well as additional references for implementation for Data Science applications	Docker	Intermediate	Familiarity with the Azure Platform, Familiarity with Server Operating Systems
		This installation and explanation explains how to use Visual Studio Code with Azure assets	VSCODE	Intermediate	Software Development, familiarity with the Microsoft Azure Platform
		This blog entry explains how to use R Studio with Microsoft R	R Studio	Intermediate	R Language experience
		This blog entry shows how to use continuous integration with Azure and GitHub	Git, GitHub	Intermediate	Software Development

Next steps

[Team Data Science Process for data scientists](#) This article provides guidance for implementing data science solutions with Azure.

Set up data science environments for use in the Team Data Science Process

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process uses various data science environments for the storage, processing, and analysis of data. They include Azure Blob Storage, several types of Azure virtual machines, HDInsight (Hadoop) clusters, and Machine Learning workspaces. The decision about which environment to use depends on the type and quantity of data to be modeled and the target destination for that data in the cloud.

- See [Quickstart: Create workspace resources you need to get started with Azure Machine Learning](#).

The **Microsoft Data Science Virtual Machine (DSVM)** is also available as an Azure virtual machine (VM) image. This VM is pre-installed and configured with several popular tools that are commonly used for data analytics and machine learning. The DSVM is available on both Windows and Linux. For more information, see [Introduction to the cloud-based Data Science Virtual Machine for Linux and Windows](#).

Learn how to create:

- [Windows DSVM](#)
- [Ubuntu DSVM](#)
- [CentOS DSVM](#)

Platforms and tools for data science projects

3/10/2022 • 8 minutes to read • [Edit Online](#)

Microsoft provides a full spectrum of analytics resources for both cloud or on-premises platforms. They can be deployed to make the execution of your data science projects efficient and scalable. Guidance for teams implementing data science projects in a trackable, version controlled, and collaborative way is provided by the [Team Data Science Process \(TDSP\)](#). See [Team Data Science Process roles and tasks](#), for an outline of the personnel roles, and their associated tasks that are handled by a data science team standardizing on this process.

The main recommended Azure resource for TDSP is [Azure Machine Learning](#). Examples in this Azure Architecture Center may show Azure Machine Learning used with other Azure resources. These other analytics resources available to data science teams using the TDSP include:

- Data Science Virtual Machines (both Windows and Linux CentOS)
- HDInsight Spark Clusters
- Azure Synapse Analytics
- Azure Data Lake
- HDInsight Hive Clusters
- Azure File Storage
- SQL Server 2019 R and Python Services
- Azure Databricks

In this document, we briefly describe the resources and provide links to the tutorials and walkthroughs the TDSP teams have published. They can help you learn how to use them step by step and start using them to build your intelligent applications. More information on these resources is available on their product pages.

Data Science Virtual Machine (DSVM)

The data science virtual machine offered on both Windows and Linux by Microsoft, contains popular tools for data science modeling and development activities. It includes tools such as:

- Microsoft R Server Developer Edition
- Anaconda Python distribution
- Jupyter notebooks for Python and R
- Visual Studio Community Edition with Python and R Tools on Windows / Eclipse on Linux
- Power BI desktop for Windows
- SQL Server 2016 Developer Edition on Windows / Postgres on Linux

It also includes **ML and AI tools** like xgboost, mxnet, and Vowpal Wabbit.

Currently DSVM is available in [Windows](#) and [Linux CentOS](#) operating systems. Choose the size of your DSVM (number of CPU cores and the amount of memory) based on the needs of the data science projects that you plan to execute on it.

For more information on Windows edition of DSVM, see [Microsoft Data Science Virtual Machine](#) on the Azure Marketplace. For the Linux edition of the DSVM, see [Linux Data Science Virtual Machine](#).

To learn how to execute some of the common data science tasks on the DSVM efficiently, see [10 things you can do on the Data science Virtual Machine](#)

Azure HDInsight Spark clusters

Apache Spark is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory computation capabilities make it a good choice for iterative algorithms in machine learning and for graph computations. Spark is also compatible with Azure Blob storage (WASB), so your existing data stored in Azure can easily be processed using Spark.

When you create a Spark cluster in HDInsight, you create Azure compute resources with Spark installed and configured. It takes about 10 minutes to create a Spark cluster in HDInsight. Store the data to be processed in Azure Blob storage. For information on using Azure Blob Storage with a cluster, see [Use HDFS-compatible Azure Blob storage with Hadoop in HDInsight](#).

TDSP team from Microsoft has published two end-to-end walkthroughs on how to use Azure HDInsight Spark Clusters to build data science solutions, one using Python and the other Scala. For more information on Azure HDInsight **Spark Clusters**, see [Overview: Apache Spark on HDInsight Linux](#). To learn how to build a data science solution using **Python** on an Azure HDInsight Spark Cluster, see [Overview of Data Science using Spark on Azure HDInsight](#). To learn how to build a data science solution using **Scala** on an Azure HDInsight Spark Cluster, see [Data Science using Scala and Spark on Azure](#).

Azure Synapse Analytics

Azure Synapse Analytics allows you to scale compute resources easily and in seconds, without over-provisioning or over-paying. It also offers the unique option to pause the use of compute resources, giving you the freedom to better manage your cloud costs. The ability to deploy scalable compute resources makes it possible to bring all your data into Azure Synapse Analytics. Storage costs are minimal and you can run compute only on the parts of datasets that you want to analyze.

For more information on Azure Synapse Analytics, see the [Azure Synapse Analytics](#) website. To learn how to build end-to-end advanced analytics solutions with Azure Synapse Analytics, see [The Team Data Science Process in action: using Azure Synapse Analytics](#).

Azure Data Lake

Azure Data Lake is as an enterprise-wide repository of every type of data collected in a single location, prior to any formal requirements, or schema being imposed. This flexibility allows every type of data to be kept in a data lake, regardless of its size or structure or how fast it is ingested. Organizations can then use Hadoop or advanced analytics to find patterns in these data lakes. Data lakes can also serve as a repository for lower-cost data preparation before curating the data and moving it into a data warehouse.

For more information on Azure Data Lake, see [Introducing Azure Data Lake](#). To learn how to build a scalable end-to-end data science solution with Azure Data Lake, see [Scalable Data Science in Azure Data Lake: An end-to-end Walkthrough](#)

Azure HDInsight Hive (Hadoop) clusters

Apache Hive is a data warehouse system for Hadoop, which enables data summarization, querying, and the analysis of data using HiveQL, a query language similar to SQL. Hive can be used to interactively explore your data or to create reusable batch processing jobs.

Hive allows you to project structure on largely unstructured data. After you define the structure, you can use Hive to query that data in a Hadoop cluster without having to use, or even know, Java or MapReduce. HiveQL (the Hive query language) allows you to write queries with statements that are similar to T-SQL.

For data scientists, Hive can run Python User-Defined Functions (UDFs) in Hive queries to process records. This

ability extends the capability of Hive queries in data analysis considerably. Specifically, it allows data scientists to conduct scalable feature engineering in languages they're mostly familiar with: the SQL-like HiveQL and Python.

For more information on Azure HDInsight Hive Clusters, see [Use Hive and HiveQL with Hadoop in HDInsight](#). To learn how to build a scalable end-to-end data science solution with Azure HDInsight Hive Clusters, see [The Team Data Science Process in action: using HDInsight Hadoop clusters](#).

Azure File Storage

Azure File Storage is a service that offers file shares in the cloud using the standard Server Message Block (SMB) Protocol. Both SMB 2.1 and SMB 3.0 are supported. With Azure File storage, you can migrate legacy applications that rely on file shares to Azure quickly and without costly rewrites. Applications running in Azure virtual machines or cloud services or from on-premises clients can mount a file share in the cloud, just as a desktop application mounts a typical SMB share. Any number of application components can then mount and access the File storage share simultaneously.

Especially useful for data science projects is the ability to create an Azure file store as the place to share project data with your project team members. Each of them then has access to the same copy of the data in the Azure file storage. They can also use this file storage to share feature sets generated during the execution of the project. If the project is a client engagement, your clients can create an Azure file storage under their own Azure subscription to share the project data and features with you. In this way, the client has full control of the project data assets. For more information on Azure File Storage, see [Get started with Azure File storage on Windows](#) and [How to use Azure File Storage with Linux](#).

SQL Server 2019 R and Python Services

R Services (In-database) provides a platform for developing and deploying intelligent applications that can uncover new insights. You can use the rich and powerful R language, including the many packages provided by the R community, to create models and generate predictions from your SQL Server data. Because R Services (In-database) integrates the R language with SQL Server, analytics are kept close to the data, which eliminates the costs and security risks associated with moving data.

R Services (In-database) supports the open source R language with a comprehensive set of SQL Server tools and technologies. They offer superior performance, security, reliability, and manageability. You can deploy R solutions using convenient and familiar tools. Your production applications can call the R runtime and retrieve predictions and visuals using Transact-SQL. You also use the ScaleR libraries to improve the scale and performance of your R solutions. For more information, see [SQL Server R Services](#).

The TDSP team from Microsoft has published two end-to-end walkthroughs that show how to build data science solutions in SQL Server 2016 R Services: one for R programmers and one for SQL developers. For **R Programmers**, see [Data Science End-to-End Walkthrough](#). For **SQL Developers**, see [In-Database Advanced Analytics for SQL Developers \(Tutorial\)](#).

Appendix: Tools to set up data science projects

Install Git Credential Manager on Windows

If you're following the TDSP on Windows, you need to install the **Git Credential Manager (GCM)** to communicate with the Git repositories. To install GCM, you first need to install Chocolatly. To install Chocolatly and the GCM, run the following commands in Windows PowerShell as an **Administrator**:

```
iwr https://chocolatey.org/install.ps1 -UseBasicParsing | iex  
choco install git-credential-manager-for-windows -y
```

Install Git on Linux (CentOS) machines

Run the following bash command to install Git on Linux (CentOS) machines:

```
sudo yum install git
```

Generate public SSH key on Linux (CentOS) machines

If you're using Linux (CentOS) machines to run the git commands, you need to add the public SSH key of your machine to your Azure DevOps services. This way the machine is recognized by the Azure DevOps Services.

First, you need to generate a public SSH key and add the key to SSH public keys in your Azure DevOps services security setting page.

1. To generate the SSH key, run the following two commands:

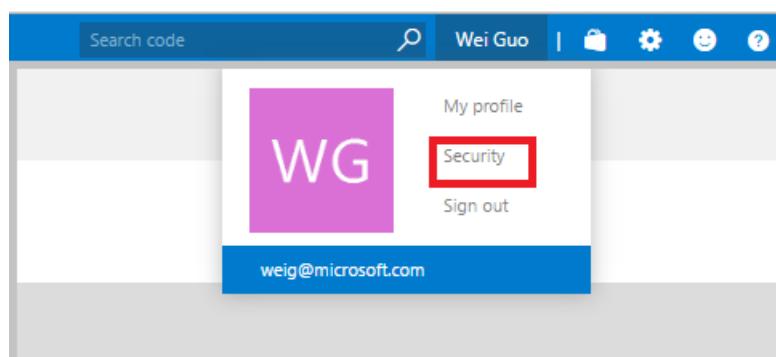
```
ssh-keygen  
cat .ssh/id_rsa.pub
```

```
[ds1@weiglinuxdsv3 ~]$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/ds1/.ssh/id_rsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/ds1/.ssh/id_rsa.  
Your public key has been saved in /home/ds1/.ssh/id_rsa.pub.  
The key fingerprint is:  
04:5d:10:2d:78:3d:e7:1b:f3:59:c9:ff:6f:92:bb:75 ds1@weiglinuxdsv3  
The key's randomart image is:  
+---[ RSA 2048]---  
* . * . |  
. S * . |  
o B o |  
. o + o .E|  
. . o + . |  
+---  
[ds1@weiglinuxdsv3 ~]$ cat .ssh/id_rsa.pub  
ssh-rsa AAAQABAAQABAAAQAC5b1xW5n1WAtAHQsqn6fCcbquP0VKq3Af135SSLgZJFpemJ3duhoBx2R80LsCgrqQVdZPgdtINC4/0Y4jhxzjd3k1w7sLU6caEl2xR1bM0fz2IN10cf1bZqGAHObt0lH9pSPR3eua/Pm2t1ZUP1MC7ch151D10K6Ms/0NsJrz454tNU+sbB925XNyvHc148hR6Uj129kOZG83SeRu69Ytgj8X2EdkpuEPxfazjSsbeauTScUFr0IEI+Jld8Pdx4LcKbYfZeENMcscIGHbrrc83JXqya05ibSMh4o4og17eb1SP18Fc5oM9PfQTlIEipzbcbvA8  
[ds1@weiglinuxdsv3 ~]$
```

2. Copy the entire ssh key including *ssh-rsa*.

3. Log in to your Azure DevOps Services.

4. Click <Your Name> at the top-right corner of the page and click **security**.



5. Click **SSH public keys**, and click **+Add**.



6. Paste the ssh key copied into the text box and save.

Next steps

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Identify scenarios and plan for advanced analytics data processing

3/10/2022 • 4 minutes to read • [Edit Online](#)

What resources are required for you to create an environment that can perform advanced analytics processing on a dataset? This article suggests a series of questions to ask that can help identify tasks and resources relevant to your scenario.

To learn about the order of high-level steps for predictive analytics, see [What is the Team Data Science Process \(TDSP\)](#). Each step requires specific resources for the tasks relevant to your particular scenario.

Answer key questions in the following areas to identify your scenario:

- data logistics
- data characteristics
- dataset quality
- preferred tools and languages

Logistic questions: data locations and movement

The logistic questions cover the following items:

- data source location
- target destination in Azure
- requirements for moving the data, including the schedule, amount, and resources involved

You may need to move the data several times during the analytics process. A common scenario is to move local data into some form of storage on Azure and then into [Azure Machine Learning](#).

What is your data source?

Is your data local or in the cloud? Possible locations include:

- a publicly available HTTP address
- a local or network file location
- a SQL Server database
- an Azure Storage container

What is the Azure destination?

Where does your data need to be for processing or modeling?

- Azure Machine Learning
- Azure Blob Storage
- SQL Azure databases
- SQL Server on Azure VM
- HDInsight (Hadoop on Azure) or Hive tables
- Mountable Azure virtual hard disks

How are you going to move the data?

For procedures and resources to ingest or load data into a variety of different storage and processing environments, see:

- Load data into storage environments for analytics
- Secure data access in Azure Machine Learning

Does the data need to be moved on a regular schedule or modified during migration?

Consider using Azure Data Factory (ADF) when data needs to be continually migrated. ADF can be helpful for:

- a hybrid scenario that involves both on-premises and cloud resources
- a scenario where the data is transacted, modified, or changed by business logic in the course of being migrated

For more information, see [Move data from a SQL Server database to SQL Azure with Azure Data Factory](#).

How much of the data is to be moved to Azure?

Large datasets may exceed the storage capacity of certain compute clusters. In such cases, you might use a sample of the data during the analysis. For details of how to down-sample a dataset in various Azure environments, see [Sample data in the Team Data Science Process](#).

Data characteristics questions: type, format, and size

These questions are key to planning your storage and processing environments. They will help you choose the appropriate scenario for your data type and understand any restrictions.

What are the data types?

- Numerical
- Categorical
- Strings
- Binary

How is your data formatted?

- Comma-separated (CSV) or tab-separated (TSV) flat files
- Compressed or uncompressed
- Azure blobs
- Hadoop Hive tables
- SQL Server tables

How large is your data?

- Small: Less than 2 GB
- Medium: Greater than 2 GB and less than 10 GB
- Large: Greater than 10 GB

As applied to Azure Machine Learning:

- [Data ingestion options for Azure Machine Learning workflows](#).
- [Optimize data processing with Azure Machine Learning](#).

Data quality questions: exploration and pre-processing

What do you know about your data?

Understand the basic characteristics about your data:

- What patterns or trends it exhibits
- What outliers it has
- How many values are missing

This step is important to help you:

- Determine how much pre-processing is needed
- Formulate hypotheses that suggest the most appropriate features or type of analysis
- Formulate plans for additional data collection

Useful techniques for data inspection include descriptive statistics calculation and visualization plots. For details of how to explore a dataset in various Azure environments, see [Explore data in the Team Data Science Process](#).

Does the data require preprocessing or cleaning?

You might need to preprocess and clean your data before you can use the dataset effectively for machine learning. Raw data is often noisy and unreliable. It might be missing values. Using such data for modeling can produce misleading results. For a description, see [Tasks to prepare data for enhanced machine learning](#).

Tools and languages questions

There are many options for languages, development environments, and tools. Be aware of your needs and preferences.

What languages do you prefer to use for analysis?

- R
- Python
- SQL
- Other

What tools could you use for data analysis?

Azure Machine Learning uses [Jupyter notebooks for data analysis](#). In addition to this recommended environment, here are other options often paired in intermediate to advanced enterprise scenarios.

- [Microsoft Azure PowerShell](#) - a script language used to administer your Azure resources in a script language
- [RStudio](#)
- [Python Tools for Visual Studio](#)
- [Microsoft Power BI](#)

Identify your advanced analytics scenario

After you have answered the questions in the previous section, you are ready to determine which scenario best fits your case. The sample scenarios are outlined in [Scenarios for advanced analytics in Azure Machine Learning](#).

Next steps

[What is the Team Data Science Process \(TDSP\)?](#)

Load data into storage environments for analytics

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process requires that data be ingested or loaded into the most appropriate way in each stage. Data destinations can include Azure Blob Storage, SQL Azure databases, SQL Server on Azure VM, HDInsight (Hadoop), Azure Synapse Analytics, and Azure Machine Learning.

The following articles describe how to ingest data into various target environments where the data is stored and processed.

- To/From [Azure Blob Storage](#)
- To [SQL Server on Azure VM](#)
- To [Azure SQL Database](#)
- To [Hive tables](#)
- To [SQL partitioned tables](#)
- From [On-premises SQL Server](#)

Technical and business needs, as well as the initial location, format, and size of your data will determine the best data ingestion plan. It is not uncommon for a best plan to have several steps. This sequence of tasks can include, for example, data exploration, pre-processing, cleaning, down-sampling, and model training. Azure Data Factory is a recommended Azure resource to orchestrate data movement and transformation.

Move data to and from Azure Blob storage

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process requires that data be ingested or loaded into a variety of different storage environments to be processed or analyzed in the most appropriate way in each stage of the process. [Azure Blob Storage has comprehensive documentation at this link](#) but this section in TDSP documentation provides a summary starter.

Different technologies for moving data

The following articles describe how to move data to and from Azure Blob storage using different technologies.

- [Azure Storage Explorer](#)
- [AzCopy](#)
- [Python](#)
- [SSIS](#)

Which method is best for you depends on your scenario. The [Scenarios for advanced analytics in Azure Machine Learning](#) article helps you determine the resources you need for a variety of data science workflows used in the advanced analytics process.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service](#).

Using Azure Data Factory

As an alternative, you can use [Azure Data Factory](#) to do the following:

- Create and schedule a pipeline that downloads data from Azure Blob storage.
- Pass it to a published Azure Machine Learning web service.
- Receive the predictive analytics results.
- Upload the results to storage.

For more information, see [Create predictive pipelines using Azure Data Factory and Azure Machine Learning](#).

Prerequisites

This article assumes that you have an Azure subscription, a storage account, and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure Storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure Storage accounts](#).

Move data to and from Azure Blob Storage using Azure Storage Explorer

3/10/2022 • 2 minutes to read • [Edit Online](#)

Azure Storage Explorer is a free tool from Microsoft that allows you to work with Azure Storage data on Windows, macOS, and Linux. This topic describes how to use it to upload and download data from Azure Blob Storage. The tool can be downloaded from [Microsoft Azure Storage Explorer](#).

This menu links to technologies you can use to move data to and from Azure Blob storage:

NOTE

If you are using VM that was set up with the scripts provided by [Data Science Virtual machines in Azure](#), then Azure Storage Explorer is already installed on the VM.

NOTE

For a complete introduction to Azure Blob Storage, refer to [Azure Blob Basics](#) and [Azure Blob Service REST API](#).

Prerequisites

This document assumes that you have an Azure subscription, a storage account, and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure Storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure Storage accounts](#). Make a note the access key for your storage account as you need this key to connect to the account with the Azure Storage Explorer tool.
- The Azure Storage Explorer tool can be downloaded from [Microsoft Azure Storage Explorer](#). Accept the defaults during install.

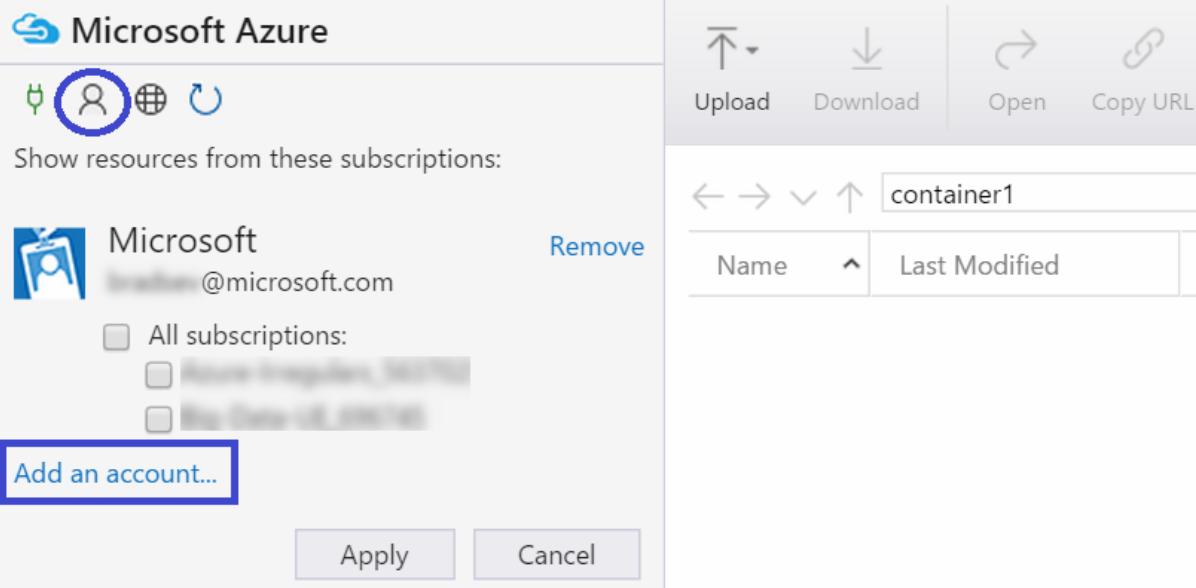
Use Azure Storage Explorer

The following steps document how to upload/download data using Azure Storage Explorer.

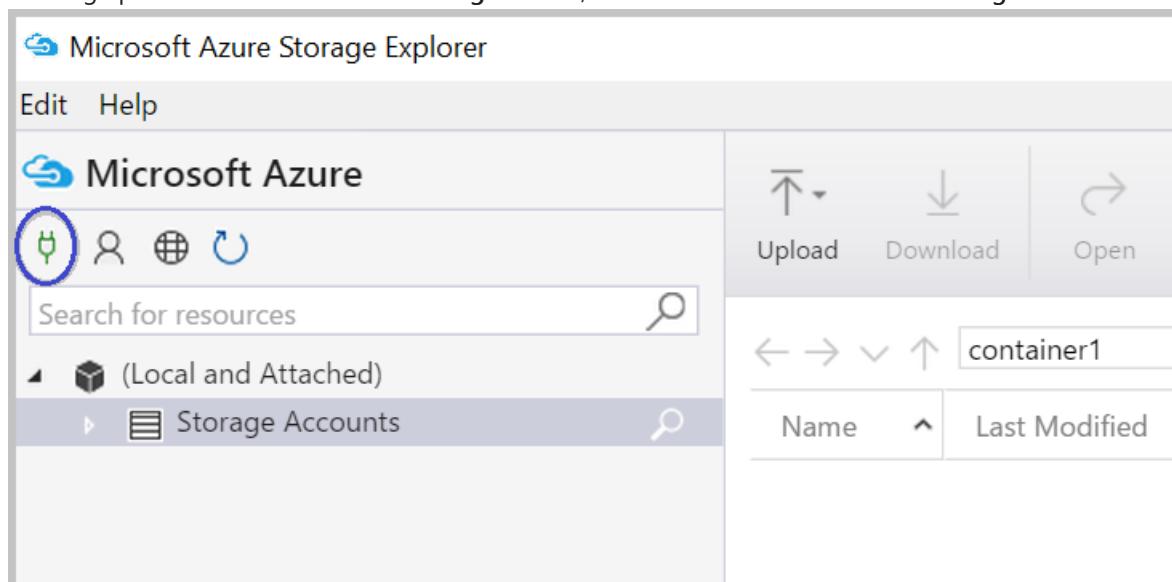
1. Launch Microsoft Azure Storage Explorer.
2. To bring up the **Sign in to your account...** wizard, select **Azure account settings** icon, then **Add an account** and enter you credentials.

Microsoft Azure Storage Explorer

Edit Help



3. To bring up the Connect to Azure Storage wizard, select the Connect to Azure Storage icon.



4. Enter the access key from your Azure Storage account on the Connect to Azure Storage wizard and then Next.

Connect to Azure Storage

Enter a connection string, Shared Access Signature (SAS) URI, or an account key.

Back

Next

Connect

Cancel

5. Enter storage account name in the **Account name** box and then select **Next**.

Attach External Storage

Enter information to connect to the Microsoft Azure storage account

Account name:

Account key:

Storage endpoints domain:

- Microsoft Azure Default
- Microsoft Azure China
- Other (specify below)

core.windows.net

Use HTTP (Not recommended)

[Online privacy statement](#)

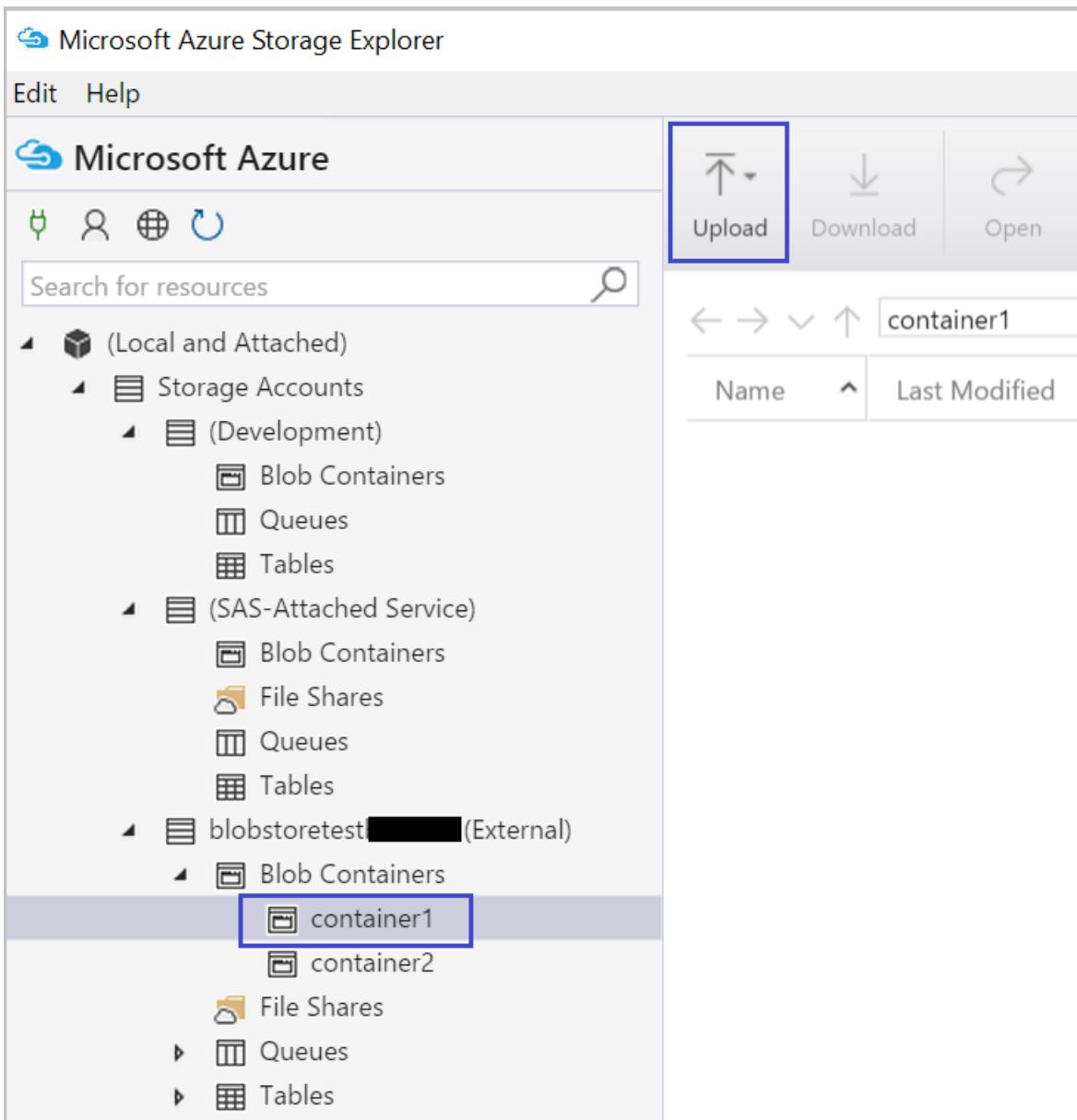
Back

Next

Connect

Cancel

6. The storage account added should now be displayed. To create a blob container in a storage account, right-click the **Blob Containers** node in that account, select **Create Blob Container**, and enter a name.
7. To upload data to a container, select the target container and click the **Upload** button.



8. Click on the ... to the right of the **Files** box, select one or multiple files to upload from the file system and click **Upload** to begin uploading the files.

Upload files

Files

No files selected



Blob type

Block Blob

Upload .vhdx files as page blobs (recommended)

Upload to folder (optional)

Upload

Cancel

9. To download data, selecting the blob in the corresponding container to download and click **Download**.

The screenshot shows the Microsoft Azure Storage Explorer interface. On the left, there's a navigation pane with sections like 'Local and Attached', 'Storage Accounts' (with 'Development' and 'SAS-Attached Service' options), and 'blobstoretestbradsev (External)' which has 'Blob Containers' (containing 'container1' and 'container2'). The main area shows a 'container1' view with a file named 'trip_data_1.csv.zip'. The 'Download' button in the toolbar above the list is highlighted with a blue box. The file 'trip_data_1.csv.zip' is also highlighted with a blue box.

Move data to or from Azure Blob Storage using SSIS connectors

3/10/2022 • 3 minutes to read • [Edit Online](#)

The [Azure Feature Pack for Integration Services \(SSIS\)](#) provides components to connect to Azure, transfer data between Azure and on-premises data sources, and process data stored in Azure.

This menu links to technologies you can use to move data to and from Azure Blob storage:

Once customers have moved on-premises data into the cloud, they can access their data from any Azure service to leverage the full power of the suite of Azure technologies. The data may be subsequently used, for example, in Azure Machine Learning or on an HDInsight cluster.

Examples for using these Azure resources are in the [SQL](#) and [HDInsight](#) walkthroughs.

For a discussion of canonical scenarios that use SSIS to accomplish business needs common in hybrid data integration scenarios, see [Doing more with SQL Server Integration Services Feature Pack for Azure](#) blog.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service REST API](#).

Prerequisites

To perform the tasks described in this article, you must have an Azure subscription and an Azure Storage account set up. You need the Azure Storage account name and account key to upload or download data.

- To set up an **Azure subscription**, see [Free one-month trial](#).
- For instructions on creating a **storage account** and for getting account and key information, see [About Azure Storage accounts](#).

To use the **SSIS connectors**, you must download:

- **SQL Server 2014 or 2016 Standard (or above)**: Install includes SQL Server Integration Services.
- **Microsoft SQL Server 2014 or 2016 Integration Services Feature Pack for Azure**: These connectors can be downloaded, respectively, from the [SQL Server 2014 Integration Services](#) and [SQL Server 2016 Integration Services](#) pages.

NOTE

SSIS is installed with SQL Server, but is not included in the Express version. For information on what applications are included in various editions of SQL Server, see [SQL Server Technical Documentation](#)

For installing SSIS, see [Install Integration Services \(SSIS\)](#)

For information on how to get up-and-running using SSIS to build simple extraction, transformation, and load (ETL) packages, see [SSIS Tutorial: Creating a Simple ETL Package](#).

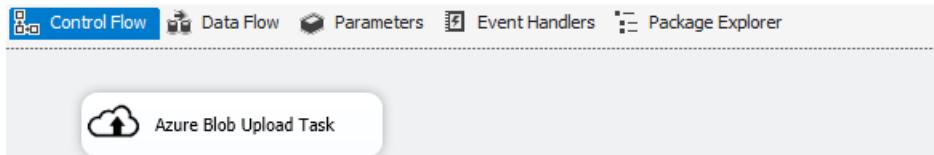
Download NYC Taxi dataset

The example described here use a publicly available dataset, either available through [Azure Open Datasets](#) or

from the source [TLC Trip Record Data](#). The dataset consists of about 173 million taxi rides in NYC in the year 2013. There are two types of data: trip details data and fare data.

Upload data to Azure blob storage

To move data using the SSIS feature pack from on-premises to Azure blob storage, we use an instance of the [Azure Blob Upload Task](#), shown here:



The parameters that the task uses are described here:

FIELD	DESCRIPTION
AzureStorageConnection	Specifies an existing Azure Storage Connection Manager or creates a new one that refers to an Azure Storage account that points to where the blob files are hosted.
BlobContainer	Specifies the name of the blob container that holds the uploaded files as blobs.
BlobDirectory	Specifies the blob directory where the uploaded file is stored as a block blob. The blob directory is a virtual hierarchical structure. If the blob already exists, it is replaced.
LocalDirectory	Specifies the local directory that contains the files to be uploaded.
FileName	Specifies a name filter to select files with the specified name pattern. For example, MySheet*.xls* includes files such as MySheet001.xls and MySheetABC.xlsx
TimeRangeFrom/TimeRangeTo	Specifies a time range filter. Files modified after <i>TimeRangeFrom</i> and before <i>TimeRangeTo</i> are included.

NOTE

The **AzureStorageConnection** credentials need to be correct and the **BlobContainer** must exist before the transfer is attempted.

Download data from Azure blob storage

To download data from Azure blob storage to on-premises storage with SSIS, use an instance of the [Azure Blob Download Task](#).

More advanced SSIS-Azure scenarios

The SSIS feature pack allows for more complex flows to be handled by packaging tasks together. For example, the blob data could feed directly into an HDInsight cluster, whose output could be downloaded back to a blob and then to on-premises storage. SSIS can run Hive and Pig jobs on an HDInsight cluster using additional SSIS connectors:

- To run a Hive script on an Azure HDInsight cluster with SSIS, use [Azure HDInsight Hive Task](#).
- To run a Pig script on an Azure HDInsight cluster with SSIS, use [Azure HDInsight Pig Task](#).

Move data to SQL Server on an Azure virtual machine

3/10/2022 • 8 minutes to read • [Edit Online](#)

This article outlines the options for moving data either from flat files (CSV or TSV formats) or from an on-premises SQL Server to SQL Server on an Azure virtual machine. These tasks for moving data to the cloud are part of the Team Data Science Process.

For a topic that outlines the options for moving data to an Azure SQL Database for Machine Learning, see [Move data to an Azure SQL Database for Azure Machine Learning](#).

The following table summarizes the options for moving data to SQL Server on an Azure virtual machine.

SOURCE	DESTINATION: SQL SERVER ON AZURE VM
Flat File	<ol style="list-style-type: none">1. Command-line bulk copy utility (BCP)2. Bulk Insert SQL Query3. Graphical Built-in Utilities in SQL Server
On-Premises SQL Server	<ol style="list-style-type: none">1. Deploy a SQL Server Database to a Microsoft Azure VM wizard2. Export to a flat File3. SQL Database Migration Wizard4. Database back up and restore

This document assumes that SQL commands are executed from SQL Server Management Studio or Visual Studio Database Explorer.

TIP

As an alternative, you can use [Azure Data Factory](#) to create and schedule a pipeline that will move data to a SQL Server VM on Azure. For more information, see [Copy data with Azure Data Factory \(Copy Activity\)](#).

Prerequisites

This tutorial assumes you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).
- An **Azure storage account**. You will use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you will need to obtain the account key used to access the storage. See [Manage storage account access keys](#).
- Provisioned **SQL Server on an Azure VM**. For instructions, see [Set up an Azure SQL Server virtual machine as an IPython Notebook server for advanced analytics](#).
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

Moving data from a flat file source to SQL Server on an Azure VM

If your data is in a flat file (arranged in a row/column format), it can be moved to SQL Server VM on Azure via

the following methods:

1. [Command-line bulk copy utility \(BCP\)](#)
2. [Bulk Insert SQL Query](#)
3. [Graphical Built-in Utilities in SQL Server \(Import/Export, SSIS\)](#)

Command-line bulk copy utility (BCP)

BCP is a command-line utility installed with SQL Server and is one of the quickest ways to move data. It works across all three SQL Server variants (On-premises SQL Server, SQL Azure, and SQL Server VM on Azure).

NOTE

Where should my data be for BCP? While it is not required, having files containing source data located on the same machine as the target SQL Server allows for faster transfers (network speed vs local disk IO speed). You can move the flat files containing data to the machine where SQL Server is installed using various file copying tools such as [AZCopy](#), [Azure Storage Explorer](#) or windows copy/paste via Remote Desktop Protocol (RDP).

1. Ensure that the database and the tables are created on the target SQL Server database. Here is an example of how to do that using the [Create Database](#) and [Create Table](#) commands:

```
CREATE DATABASE <database_name>

CREATE TABLE <tablename>
(
    <columnname1> <datatype> <constraint>,
    <columnname2> <datatype> <constraint>,
    <columnname3> <datatype> <constraint>
)
```

2. Generate the format file that describes the schema for the table by issuing the following command from the command line of the machine where bcp is installed.

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -S servername\sqlinstance -T -t \t  
-r \n
```

3. Insert the data into the database using the bcp command, which should work from the command line when SQL Server is installed on same machine:

```
bcp dbname..tablename in datafilename.tsv -f exportformatfilename.xml -S servername\sqlinstancename -U  
username -P password -b block_size_to_move_in_single_attempt -t \t -r \n
```

Optimizing BCP Inserts Please refer the following article '[Guidelines for Optimizing Bulk Import](#)' to optimize such inserts.

Parallelizing Inserts for Faster Data Movement

If the data you are moving is large, you can speed up things by simultaneously executing multiple BCP commands in parallel in a PowerShell Script.

NOTE

Big data Ingestion To optimize data loading for large and very large datasets, partition your logical and physical database tables using multiple file groups and partition tables. For more information about creating and loading data to partition tables, see [Parallel Load SQL Partition Tables](#).

The following sample PowerShell script demonstrates parallel inserts using bcp:

```

$NO_OF_PARALLEL_JOBS=2

Set-ExecutionPolicy RemoteSigned #set execution policy for the script to execute
# Define what each job does
$ScriptBlock = {
    param($partitionnumber)

    #Explicitly using SQL username password
    bcp database..tablename in datafile_path.csv -F 2 -f format_file_path.xml -U username@servername -S
    tcp:servername -P password -b block_size_to_move_in_single_attempt -t "," -r \n -o
    path_to_outputfile.$partitionnumber.txt

    #Trusted connection w/o username password (if you are using windows auth and are signed in with that
    #credentials)
    #bcp database..tablename in datafile_path.csv -o path_to_outputfile.$partitionnumber.txt -h "TABLOCK" -F
    2 -f format_file_path.xml -T -b block_size_to_move_in_single_attempt -t "," -r \n
}

# Background processing of all partitions
for ($i=1; $i -le $NO_OF_PARALLEL_JOBS; $i++)
{
    Write-Debug "Submit loading partition # $i"
    Start-Job $ScriptBlock -Arg $i
}

# Wait for it all to complete
While (Get-Job -State "Running")
{
    Start-Sleep 10
    Get-Job
}

# Getting the information back from the jobs
Get-Job | Receive-Job
Set-ExecutionPolicy Restricted #reset the execution policy

```

Bulk Insert SQL Query

[Bulk Insert SQL Query](#) can be used to import data into the database from row/column based files (the supported types are covered in the[Prepare Data for Bulk Export or Import \(SQL Server\)](#) topic).

Here are some sample commands for Bulk Insert are as below:

1. Analyze your data and set any custom options before importing to make sure that the SQL Server database assumes the same format for any special fields such as dates. Here is an example of how to set the date format as year-month-day (if your data contains the date in year-month-day format):

```
SET DATEFORMAT ymd;
```

2. Import data using bulk import statements:

```

BULK INSERT <tablename>
FROM
'<datafilename>'
WITH
(
    FirstRow = 2,
    FIELDTERMINATOR = ',', --this should be column separator in your data
    ROWTERMINATOR = '\n'   --this should be the row separator in your data
)

```

Built-in Utilities in SQL Server

You can use SQL Server Integration Services (SSIS) to import data into SQL Server VM on Azure from a flat file. SSIS is available in two studio environments. For details, see [Integration Services \(SSIS\) and Studio Environments](#):

- For details on SQL Server Data Tools, see [Microsoft SQL Server Data Tools](#)
- For details on the Import/Export Wizard, see [SQL Server Import and Export Wizard](#)

Moving Data from on-premises SQL Server to SQL Server on an Azure VM

You can also use the following migration strategies:

1. [Deploy a SQL Server Database to a Microsoft Azure VM wizard](#)
2. [Export to Flat File](#)
3. [SQL Server Migration Assistant \(SSMA\)](#)
4. [Database back up and restore](#)

We describe each of these options below:

Deploy a SQL Server Database to a Microsoft Azure VM wizard

The [Deploy a SQL Server Database to a Microsoft Azure VM wizard](#) is a simple and recommended way to move data from an on-premises SQL Server instance to SQL Server on an Azure VM. For detailed steps as well as a discussion of other alternatives, see [Migrate a database to SQL Server on an Azure VM](#).

Export to Flat File

Various methods can be used to bulk export data from an On-Premises SQL Server as documented in the [Bulk Import and Export of Data \(SQL Server\)](#) topic. This document will cover the Bulk Copy Program (BCP) as an example. Once data is exported into a flat file, it can be imported to another SQL server using bulk import.

1. Export the data from on-premises SQL Server to a file using the bcp utility as follows

```
bcp dbname..tablename out datafile.tsv -S servername\sqlinstancename -T -t \t -t \n -c
```

2. Create the database and the table on SQL Server VM on Azure using the `create database` and `create table` for the table schema exported in step 1.

3. Create a format file for describing the table schema of the data being exported/imported. Details of the format file are described in [Create a Format File \(SQL Server\)](#).

Format file generation when running BCP from the SQL Server computer

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -S servername\sqlinstance -T -t \t -r \n
```

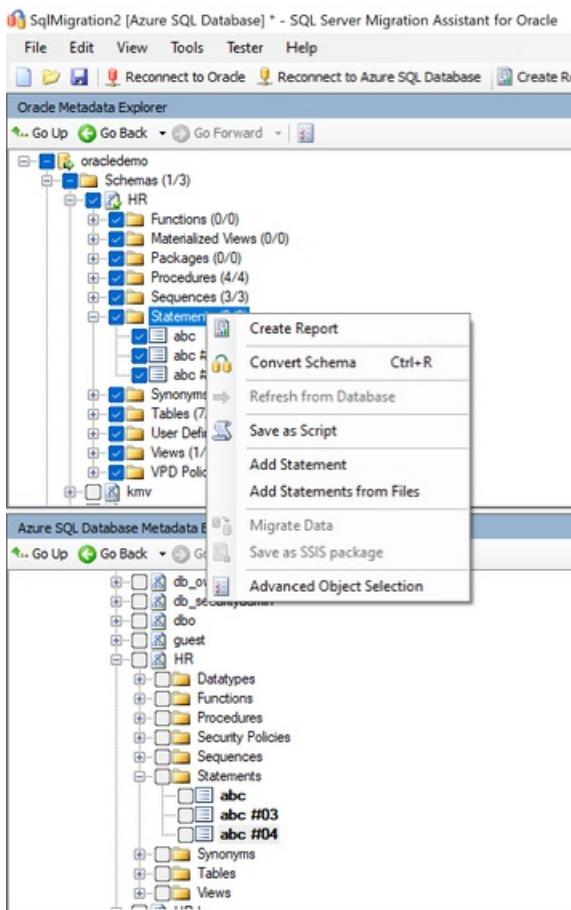
Format file generation when running BCP remotely against a SQL Server

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -U  
username@servername.database.windows.net -S tcp:servername -P password --t \t -r \n
```

4. Use any of the methods described in section [Moving Data from File Source](#) to move the data in flat files to a SQL Server.

SQL Server Migration Assistant (SSMA)

[SQL Server Migration Assistant \(SSMA\)](#) provides a user-friendly way to move data between two SQL server instances. It allows the user to map the data schema between sources and destination tables, choose column types and various other functionalities. It uses bulk copy (BCP) under the covers. A screenshot of the welcome screen for SQL Server Migration Assistant (SSMA) is shown below.



Database back up and restore

SQL Server supports:

- Database back up and restore functionality** (both to a local file or bacpac export to blob) and [Data Tier Applications](#) (using bacpac).
- Ability to directly create SQL Server VMs on Azure with a copied database or copy to an existing database in SQL Database. For more information, see [Use the Copy Database Wizard](#).

A screenshot of the Database back up/restore options from SQL Server Management Studio is shown below.

objid	objid_fanularity	objid_holiness	objid_id	objid_latitude	objid_longitude	objid_mbid	objid_vblog
0.639902515496	0.461318337541	APRNUU112298900C91	0	NULL	0	Eae6a016-91d7-48cc-ba7d-5e0a5d320c54	NULL
0.6717550353	0.386663634086	APR73MO11878940578	37.77916	San Francisco, CA	-122.42005	911ce6e6-6338-4ab9-a5d3-600964b1453	NULL
0.581793705945	0.401950543386	APRD7TVE11078999FB1	0	California - LA	0	e77e5f5d-4761-45b5-9347-2051811e366	NULL
0.47694091237	0.30708016881	APR73MO11878940578	33.62574	Mexico	-101.95625	a162c842-413-4395-9d62-29569e0d4344	NULL
0.63063003759	0.417458844571	APRMABH110789546F3	35.14968	New Jersey	0	1c7ba6c6-db33-4433-940b-78d0ff1949c2	electric pop and ro
0.00131579730989	0.37932374320	APR8Q9/C11879756098	0	Memphis, TN	-90.04892	c528949-7854-4419-a935-736387678781	NULL
0.93134211069	0.33828861594	APRQ286111878960P3	64.55663	Chennai, SC	12.66953	c795a5d-7327-4375-638-686141622	NULL
0.627827825331	0.389716850902	APRC736118789423D	37.77916	London	-0.12714	3e75a3b-e25a-4408-9949-e56f5c53be1	NULL
0.4212993845	0.336665560682	APRNDSW111878939DF	51.50632	San Francisco, CA	-122.42005	c1632a11-027c-44a4-a4fc-ba5f5e292005	uk_bathengtshp
0.487296790928	0.343428378297	APKRUR111878984D4	0	NULL	0	7e273884-ed9-4451-9c4d-39b3803805ebcd	NULL

Resources

[Migrate a Database to SQL Server on an Azure VM](#)

[SQL Server on Azure Virtual Machines overview](#)

Move data to Azure SQL Database for Azure Machine Learning

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article outlines the options for moving data either from flat files (CSV or TSV formats) or from data stored in SQL Server to an Azure SQL Database. These tasks for moving data to the cloud are part of the Team Data Science Process.

For a topic that outlines the options for migrating data from SQL Server into Azure SQL options, see [Migrate to Azure SQL](#).

The following table summarizes the options for moving data to an Azure SQL Database.

SOURCE	DESTINATION: AZURE SQL
Flat file (CSV or TSV formatted)	Bulk Insert SQL Query
On-premises SQL Server	1. Export to Flat File 2. SQL Server Migration Assistant (SSMA) 3. Database back up and restore 4. Azure Data Factory

Prerequisites

The procedures outlined here require that you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).
- An **Azure storage account**. You use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you need to obtain the account key used to access the storage. See [Manage storage account access keys](#).
- Access to an **Azure SQL Database**. If you must set up an Azure SQL Database, [Getting Started with Microsoft Azure SQL Database](#) provides information on how to provision a new instance of an Azure SQL Database.
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

Data: The migration processes are demonstrated using the [NYC Taxi dataset](#). The NYC Taxi dataset contains information on trip data and fares, which is either available through [Azure Open Datasets](#) or from the source [TLC Trip Record Data](#). A sample and description of these files are provided in [NYC Taxi Trips Dataset Description](#).

You can either adapt the procedures described here to a set of your own data or follow the steps as described by using the NYC Taxi dataset. To upload the NYC Taxi dataset into your SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server Database](#).

Moving data from a flat file source to an Azure SQL Database

Data in flat files (CSV or TSV formatted) can be moved to an Azure SQL Database using a Bulk Insert SQL Query.

Bulk Insert SQL Query

The steps for the procedure using the Bulk Insert SQL Query are similar to the directions for moving data from a flat file source to SQL Server on an Azure VM. For details, see [Bulk Insert SQL Query](#).

Moving Data from SQL Server to an Azure SQL Database

If the source data is stored in SQL Server, there are various possibilities for moving the data to an Azure SQL Database:

1. [Export to Flat File](#)
2. [SQL Server Migration Assistant \(SSMA\)](#)
3. [Database back up and restore](#)
4. [Azure Data Factory](#)

The steps for the first three are similar to those sections in [Move data to SQL Server on an Azure virtual machine](#) that cover these same procedures. Links to the appropriate sections in that topic are provided in the following instructions.

Export to Flat File

The steps for this exporting to a flat file are similar to those directions covered in [Export to Flat File](#).

SQL Server Migration Assistant (SSMA)

The steps for using the SQL Server Migration Assistant (SSMA) are similar to those directions covered in [SQL Server Migration Assistant \(SSMA\)](#).

Database back up and restore

The steps for using database backup and restore are similar to those directions listed in [Database backup and restore](#).

Azure Data Factory

Learn how to move data to an Azure SQL Database with Azure Data Factory (ADF) in this topic, [Move data from a SQL Server to SQL Azure with Azure Data Factory](#). This topic shows how to use ADF to move data from a SQL Server database to an Azure SQL Database via Azure Blob Storage.

Consider using ADF when data needs to be continually migrated with hybrid on-premises and cloud sources. ADF also helps when the data needs transformations, or needs new business logic during migration. ADF allows for the scheduling and monitoring of jobs using simple JSON scripts that manage the movement of data on a periodic basis. ADF also has other capabilities such as support for complex operations.

Create Hive tables and load data from Azure Blob Storage

3/10/2022 • 10 minutes to read • [Edit Online](#)

This article presents generic Hive queries that create Hive tables and load data from Azure Blob Storage. Some guidance is also provided on partitioning Hive tables and on using the Optimized Row Columnar (ORC) formatting to improve query performance.

Prerequisites

This article assumes that you have:

- Created an Azure Storage account. If you need instructions, see [About Azure Storage accounts](#).
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Setup Clusters in HDInsight](#).
- Enabled remote access to the cluster, logged in, and opened the Hadoop Command-Line console. If you need instructions, see [Manage Apache Hadoop clusters](#).

Upload data to Azure Blob Storage

If you created an Azure virtual machine by following the instructions provided in [Set up an Azure virtual machine for advanced analytics](#), this script file should have been downloaded to the *C:\Users\<user name>\Documents\Data Science Scripts* directory on the virtual machine. These Hive queries only require that you provide a data schema and Azure Blob Storage configuration in the appropriate fields to be ready for submission.

We assume that the data for Hive tables is in an **uncompressed** tabular format, and that the data has been uploaded to the default (or to an additional) container of the storage account used by the Hadoop cluster.

If you want to practice on the [NYC Taxi Trip Data](#), you need to:

- **download** the 24 NYC Taxi Trip Data files (12 Trip files and 12 Fare files) -- either available through [Azure Open Datasets](#) or from the source [TLC Trip Record Data](#),
- **unzip** all files into .csv files, and then
- **upload** them to the default (or appropriate container) of the Azure Storage account; options for such an account appear at [Use Azure Storage with Azure HDInsight clusters](#) topic. The process to upload the .csv files to the default container on the storage account can be found on this [page](#).

How to submit Hive queries

Hive queries can be submitted by using:

- [Submit Hive queries through Hadoop Command Line in headnode of Hadoop cluster](#)
- [Submit Hive queries with the Hive Editor](#)
- [Submit Hive queries with Azure PowerShell Commands](#)

Hive queries are SQL-like. If you are familiar with SQL, you may find the [Hive for SQL Users Cheat Sheet](#) useful.

When submitting a Hive query, you can also control the destination of the output from Hive queries, whether it be on the screen or to a local file on the head node or to an Azure blob.

Submit Hive queries through Hadoop Command Line in headnode of Hadoop cluster

If the Hive query is complex, submitting it directly in the head node of the Hadoop cluster typically leads to faster turn around than submitting it with a Hive Editor or Azure PowerShell scripts.

Log in to the head node of the Hadoop cluster, open the Hadoop Command Line on the desktop of the head node, and enter command `cd %hive_home%\bin`.

You have three ways to submit Hive queries in the Hadoop Command Line:

- directly
- using `.hql` files
- with the Hive command console

Submit Hive queries directly in Hadoop Command Line.

You can run command like `hive -e "<your hive query>"` to submit simple Hive queries directly in Hadoop Command Line. Here is an example, where the red box outlines the command that submits the Hive query, and the green box outlines the output from the Hive query.

The screenshot shows a Windows command prompt window titled "Hadoop Command Line". The command entered is `hive -e "show tables;"`. The output shows the results of the query, which is a single row: `hivesampletable`. The entire command line and its output are highlighted with a red box, while the specific query result is highlighted with a green box.

```
C:\apps\dist\hadoop-2.4.0.2.1.10.0-2290>hive -e "show tables;"  
hive is not recognized as an internal or external command,  
operable program or batch file.  
C:\apps\dist\hadoop-2.4.0.2.1.10.0-2290>cd %hive_home%\bin  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "show tables;"  
Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.  
.0-2290/conf/hive-log4j.properties  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/  
hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j.impl.StaticLoggerBinder.cl  
ass]  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoo  
p2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j.impl.StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
ok  
hivesampletable  
Time taken: 1.502 seconds. Fetched: 1 row(s)  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>
```

Submit Hive queries in `.hql` files

When the Hive query is more complicated and has multiple lines, editing queries in command line or Hive command console is not practical. An alternative is to use a text editor in the head node of the Hadoop cluster to save the Hive queries in an `.hql` file in a local directory of the head node. Then the Hive query in the `.hql` file can be submitted by using the `-f` argument as follows:

```
hive -f "<path to the .hql file>"
```

The screenshot shows a Windows command prompt window titled "Hadoop Command Line". The command entered is `hive -f "C:/apps/temp/hivequeryinfile.hql"`. The output shows the results of the query, which is a list of states and their abbreviations. The entire command line and its output are highlighted with a red box, while the specific query result is highlighted with a green box.

```
directory does not exist. Enter new>  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290>hive -f "C:/apps/temp/hivequeryinfile.hql"  
Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.  
.0-2290/conf/hive-log4j.properties  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/  
hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j.impl.StaticLoggerBinder.cl  
ass]  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoo  
p2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j.impl.StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
ok  
select  
from hivesampletable  
limit 10;  
United States 18:54:28 aa-00 Android Samsung SCH-I500 California  
United States 13.9704007 -00 Android HTC Incredible Pennsylvania  
United States 11.111111 -00 Android HTC Incredible Pennsylvania  
United States 1.475762 -00 Android HTC Incredible Pennsylvania  
United States 0.245968 -00 Android HTC Incredible Pennsylvania  
United States 20.3095377 -00 Android Motorola Droid R Colorado  
United States 16.2901668 -00 Android Motorola Droid X Colorado  
United States 1.7795228 -00 Android Motorola Droid X Utah United  
States 18.0441247 -00 Android Motorola Droid X Utah United  
States 11.7451457 -00 Android Motorola Droid X Utah United  
States 0.6177179 -00 Android Motorola Droid X Colorado  
United States 18.0441247 -00 Android Motorola Droid X Colorado  
Time taken: 2.995 seconds. Fetched: 10 rows<  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290>
```

SUPPRESS PROGRESS STATUS SCREEN PRINT OF HIVE QUERIES

By default, after Hive query is submitted in Hadoop Command Line, the progress of the Map/Reduce job is printed out on screen. To suppress the screen print of the Map/Reduce job progress, you can use an argument `-S` ("S" in upper case) in the command line as follows:

```
hive -S -f "<path to the .hql file>"  
hive -S -e "<Hive queries>"
```

Submit Hive queries in Hive command console.

You can also first enter the Hive command console by running command `hive` in Hadoop Command Line, and then submit Hive queries in Hive command console. Here is an example. In this example, the two red boxes highlight the commands used to enter the Hive command console, and the Hive query submitted in Hive command console, respectively. The green box highlights the output from the Hive query.

The screenshot shows a Windows command prompt window titled "Hadoop Command Line - hive". The command `hive` is entered, followed by a red box highlighting the command `select * from hivesampletable limit 3;`. The output of the query, which displays three rows of data, is highlighted with a green box. The data is as follows:

	18:54:20	en-US	Android	Samsung	SCH-i500	California
United States	13.9204007	0	0			
23	19:19:44	en-US	Android	HTC	Incredible	Pennsylvania
United States	NULL	0	0			
23	19:19:46	en-US	Android	HTC	Incredible	Pennsylvania
United States	1.4757422	0	1			

The previous examples directly output the Hive query results on screen. You can also write the output to a local file on the head node, or to an Azure blob. Then, you can use other tools to further analyze the output of Hive queries.

Output Hive query results to a local file. To output Hive query results to a local directory on the head node, you have to submit the Hive query in the Hadoop Command Line as follows:

```
hive -e "<hive query>" > <local path in the head node>
```

In the following example, the output of Hive query is written into a file `hivequeryoutput.txt` in directory

```
C:\apps\temp
```

The screenshot shows the Hadoop Command Line window and a File Explorer window. The command `hive -e "select * from hivesampletable limit 3;" > C:\apps\temp\hivequeryoutput.txt` is run in the command line, and the resulting file `hivequeryoutput.txt` is shown in the File Explorer under the `temp` directory. The file contains the same data as the previous screenshot.

Output Hive query results to an Azure blob

You can also output the Hive query results to an Azure blob, within the default container of the Hadoop cluster. The Hive query for this is as follows:

```
insert overwrite directory wasb:///<directory within the default container> <select clause from ...>
```

In the following example, the output of Hive query is written to a blob directory `queryoutputdir` within the default container of the Hadoop cluster. Here, you only need to provide the directory name, without the blob name. An error is thrown if you provide both directory and blob names, such as

```
wasb:///queryoutputdir/queryoutput.txt.
```



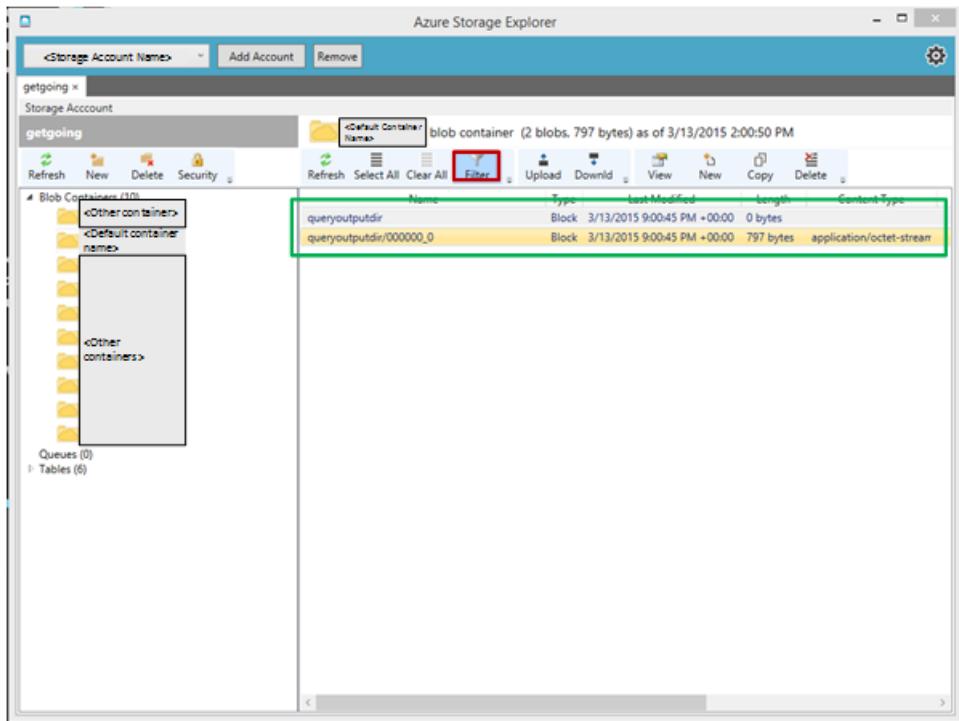
The screenshot shows a Windows command-line interface titled "Hadoop Command Line". The command entered is:

```
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "insert overwrite directory 'wasb:///queryoutputdir' select * from hivesamptable limit 10;"
```

The output of the command is displayed below the command line. It shows the configuration of the Hadoop environment, the execution of the Hive job, and the completion of the job. The output ends with:

```
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>
```

If you open the default container of the Hadoop cluster using Azure Storage Explorer, you can see the output of the Hive query as shown in the following figure. You can apply the filter (highlighted by red box) to only retrieve the blob with specified letters in names.



Submit Hive queries with the Hive Editor

You can also use the Query Console (Hive Editor) by entering a URL of the form `https://<Hadoop cluster name>.azurehdinsight.net/Home/HiveEditor` into a web browser. You must be logged in to see this console and so you need your Hadoop cluster credentials here.

Submit Hive queries with Azure PowerShell Commands

You can also use PowerShell to submit Hive queries. For instructions, see [Submit Hive jobs using PowerShell](#).

Create Hive database and tables

The Hive queries are shared in the [GitHub repository](#) and can be downloaded from there.

Here is the Hive query that creates a Hive table.

```
create database if not exists <database name>;
CREATE EXTERNAL TABLE if not exists <database name>.<table name>
(
    field1 string,
    field2 int,
    field3 float,
    field4 double,
    ...,
    fieldN string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' lines terminated by '<line separator>'
STORED AS TEXTFILE LOCATION '<storage location>' TBLPROPERTIES("skip.header.line.count"="1");
```

Here are the descriptions of the fields that you need to plug in and other configurations:

- **<database name>**: the name of the database that you want to create. If you just want to use the default database, the query "create database..." can be omitted.
- **<table name>**: the name of the table that you want to create within the specified database. If you want to use the default database, the table can be directly referred by <table name> without <database name>.
- **<field separator>**: the separator that delimits fields in the data file to be uploaded to the Hive table.
- **<line separator>**: the separator that delimits lines in the data file.
- **<storage location>**: the Azure Storage location to save the data of Hive tables. If you do not specify

LOCATION <storage location>, the database and the tables are stored in *hive/warehouse*/directory in the default container of the Hive cluster by default. If you want to specify the storage location, the storage location has to be within the default container for the database and tables. This location has to be referred as location relative to the default container of the cluster in the format of '*wasb:///<directory 1>/*' or '*wasb:///<directory 1>/<directory 2>/*', etc. After the query is executed, the relative directories are created within the default container.

- **TBLPROPERTIES("skip.header.line.count" = "1")**: If the data file has a header line, you have to add this property **at the end** of the *create table* query. Otherwise, the header line is loaded as a record to the table. If the data file does not have a header line, this configuration can be omitted in the query.

Load data to Hive tables

Here is the Hive query that loads data into a Hive table.

```
LOAD DATA INPATH '<path to blob data>' INTO TABLE <database name>.<table name>;
```

- **<path to blob data>**: If the blob file to be uploaded to the Hive table is in the default container of the HDInsight Hadoop cluster, the *<path to blob data>* should be in the format '*wasb://<directory in this container>/<blob file name>*'. The blob file can also be in an additional container of the HDInsight Hadoop cluster. In this case, *<path to blob data>* should be in the format '*wasb://<container name>@<storage account name>.blob.core.windows.net/<blob file name>*'.

NOTE

The blob data to be uploaded to Hive table has to be in the default or additional container of the storage account for the Hadoop cluster. Otherwise, the *LOAD DATA* query fails complaining that it cannot access the data.

Advanced topics: partitioned table and store Hive data in ORC format

If the data is large, partitioning the table is beneficial for queries that only need to scan a few partitions of the table. For instance, it is reasonable to partition the log data of a web site by dates.

In addition to partitioning Hive tables, it is also beneficial to store the Hive data in the Optimized Row Columnar (ORC) format. For more information on ORC formatting, see [Using ORC files improves performance when Hive is reading, writing, and processing data](#).

Partitioned table

Here is the Hive query that creates a partitioned table and loads data into it.

```
CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<table name>
  (field1 string,
  ...
  fieldN string
)
PARTITIONED BY (<partitionfieldname> vartype) ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
  lines terminated by '<line separator>' TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<partitioned table name>
  PARTITION (<partitionfieldname>=<partitionfieldvalue>);
```

When querying partitioned tables, it is recommended to add the partition condition in the **beginning** of the `where` clause, which improves the search efficiency.

```

select
    field1, field2, ..., fieldN
from <database name>.<partitioned table name>
where <partitionfieldname>=<partitionfieldvalue> and ...;
```

Store Hive data in ORC format

You cannot directly load data from blob storage into Hive tables that is stored in the ORC format. Here are the steps that you need to take to load data from Azure blobs to Hive tables stored in ORC format.

Create an external table **STORED AS TEXTFILE** and load data from blob storage to the table.

```

CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<external textfile table name>
(
    field1 string,
    field2 int,
    ...
    fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
lines terminated by '<line separator>' STORED AS TEXTFILE
LOCATION 'wasb:///<directory in Azure blob>' TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<table name>;
```

Create an internal table with the same schema as the external table in step 1, with the same field delimiter, and store the Hive data in the ORC format.

```

CREATE TABLE IF NOT EXISTS <database name>.<ORC table name>
(
    field1 string,
    field2 int,
    ...
    fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' STORED AS ORC;
```

Select data from the external table in step 1 and insert into the ORC table

```

INSERT OVERWRITE TABLE <database name>.<ORC table name>
SELECT * FROM <database name>.<external textfile table name>;
```

NOTE

If the TEXTFILE table *<database name>.<external textfile table name>* has partitions, in STEP 3, the `SELECT * FROM <database name>.<external textfile table name>` command selects the partition variable as a field in the returned data set. Inserting it into the *<database name>.<ORC table name>* fails since *<database name>.<ORC table name>* does not have the partition variable as a field in the table schema. In this case, you need to specifically select the fields to be inserted to *<database name>.<ORC table name>* as follows:

```

INSERT OVERWRITE TABLE <database name>.<ORC table name> PARTITION (<partition variable>=<partition value>)
    SELECT field1, field2, ..., fieldN
    FROM <database name>.<external textfile table name>
    WHERE <partition variable>=<partition value>;
```

It is safe to drop the *<external text file table name>* when using the following query after all data has been

inserted into <database name>.<ORC table name>:

```
DROP TABLE IF EXISTS <database name>.<external textfile table name>;
```

After following this procedure, you should have a table with data in the ORC format ready to use.

Build and optimize tables for fast parallel import of data into SQL Server on an Azure VM

3/10/2022 • 5 minutes to read • [Edit Online](#)

This article describes how to build partitioned tables for fast parallel bulk importing of data to a SQL Server database. For big data loading/transfer to a SQL database, importing data to the SQL database and subsequent queries can be improved by using *Partitioned Tables and Views*.

Create a new database and a set of filegroups

- [Create a new database](#), if it doesn't exist already.
- Add database filegroups to the database, which holds the partitioned physical files.
- This can be done with [CREATE DATABASE](#) if new or [ALTER DATABASE](#) if the database exists already.
- Add one or more files (as needed) to each database filegroup.

NOTE

Specify the target filegroup, which holds data for this partition and the physical database file name(s) where the filegroup data is stored.

The following example creates a new database with three filegroups other than the primary and log groups, containing one physical file in each. The database files are created in the default SQL Server Data folder, as configured in the SQL Server instance. For more information about the default file locations, see [File Locations for Default and Named Instances of SQL Server](#).

```
DECLARE @data_path nvarchar(256);
SET @data_path = (SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
    FROM master.sys.master_files
    WHERE database_id = 1 AND file_id = 1);

EXECUTE (
    CREATE DATABASE <database_name>
        ON PRIMARY
            ( NAME = 'Primary' , FILENAME = '' + @data_path + '<primary_file_name>.mdf' , SIZE = 4096KB ,
FILEGROWTH = 1024KB ),
        FILEGROUP [filegroup_1]
            ( NAME = 'FileGroup1' , FILENAME = '' + @data_path + '<file_name_1>.ndf' , SIZE = 4096KB ,
FILEGROWTH = 1024KB ),
        FILEGROUP [filegroup_2]
            ( NAME = 'FileGroup2' , FILENAME = '' + @data_path + '<file_name_2>.ndf' , SIZE = 4096KB ,
FILEGROWTH = 1024KB ),
        FILEGROUP [filegroup_3]
            ( NAME = 'FileGroup3' , FILENAME = '' + @data_path + '<file_name_3>.ndf' , SIZE = 102400KB ,
FILEGROWTH = 10240KB )
        LOG ON
            ( NAME = 'LogFileGroup' , FILENAME = '' + @data_path + '<log_file_name>.ldf' , SIZE = 1024KB ,
FILEGROWTH = 10%)
    )
```

Create a partitioned table

To create partitioned table(s) according to the data schema, mapped to the database filegroups created in the previous step, you must first create a partition function and scheme. When data is bulk imported to the partitioned table(s), records are distributed among the filegroups according to a partition scheme, as described below.

1. Create a partition function

[Create a partition function](#) This function defines the range of values/boundaries to be included in each individual partition table, for example, to limit partitions by month(some_datetime_field) in the year 2013:

```
CREATE PARTITION FUNCTION <DatetimeFieldPFN>(<datetime_field>)
AS RANGE RIGHT FOR VALUES (
    '20130201', '20130301', '20130401',
    '20130501', '20130601', '20130701', '20130801',
    '20130901', '20131001', '20131101', '20131201' )
```

2. Create a partition scheme

[Create a partition scheme](#). This scheme maps each partition range in the partition function to a physical filegroup, for example:

```
CREATE PARTITION SCHEME <DatetimeFieldPScheme> AS
    PARTITION <DatetimeFieldPFN> TO (
        <filegroup_1>, <filegroup_2>, <filegroup_3>, <filegroup_4>,
        <filegroup_5>, <filegroup_6>, <filegroup_7>, <filegroup_8>,
        <filegroup_9>, <filegroup_10>, <filegroup_11>, <filegroup_12> )
```

To verify the ranges in effect in each partition according to the function/scheme, run the following query:

```
SELECT psch.name as PartitionScheme,
       prng.value AS PartitionValue,
       prng.boundary_id AS BoundaryID
  FROM sys.partition_functions AS pfun
 INNER JOIN sys.partition_schemes psch ON pfun.function_id = psch.function_id
 INNER JOIN sys.partition_range_values prng ON prng.function_id=pfun.function_id
 WHERE pfun.name = <DatetimeFieldPFN>
```

3. Create a partition table

[Create partitioned table\(s\)](#) according to your data schema, and specify the partition scheme and constraint field used to partition the table, for example:

```
CREATE TABLE <table_name> ( [include schema definition here] )
    ON <TablePScheme>(<partition_field>)
```

For more information, see [Create Partitioned Tables and Indexes](#).

Bulk import the data for each individual partition table

- You may use BCP, BULK INSERT, or other methods such as [Microsoft Data Migration](#). The example provided uses the BCP method.
- [Alter the database](#) to change transaction logging scheme to BULK_LOGGED to minimize overhead of logging, for example:

```
ALTER DATABASE <database_name> SET RECOVERY BULK_LOGGED
```

- To expedite data loading, launch the bulk import operations in parallel. For tips on expediting bulk importing of big data into SQL Server databases, see [Load 1 TB in less than 1 hour](#).

The following PowerShell script is an example of parallel data loading using BCP.

```

# Set database name, input data directory, and output log directory
# This example loads comma-separated input data files
# The example assumes the partitioned data files are named as <base_file_name>_<partition_number>.csv
# Assumes the input data files include a header line. Loading starts at line number 2.

$dbname = "<database_name>"
$indir = "<path_to_data_files>"
$logdir = "<path_to_log_directory>"

# Select authentication mode
$sqlauth = 0

# For SQL authentication, set the server and user credentials
$sqlusr = "<user@server>"
$server = "<tcp:serverdns>"
$pass = "<password>"

# Set number of partitions per table - Should match the number of input data files per table
$numofparts = <number_of_partitions>

# Set table name to be loaded, basename of input data files, input format file, and number of partitions
$tbname = "<table_name>"
$basename = "<base_input_data_filename_no_extension>"
$fmtfile = "<full_path_to_format_file>"

# Create log directory if it does not exist
New-Item -ErrorAction Ignore -ItemType directory -Path $logdir

# BCP example using Windows authentication
$ScriptBlock1 = {
    param($dbname, $tbname, $basename, $fmtfile, $indir, $logdir, $num)
    bcp ($dbname + ".." + $tbname) in ($indir + "\\" + $basename + "_" + $num + ".csv") -o ($logdir + "\\" +
    $tbname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($fmtfile) -T -b 2500 -t "," -r \n
}

# BCP example using SQL authentication
$ScriptBlock2 = {
    param($dbname, $tbname, $basename, $fmtfile, $indir, $logdir, $num, $sqlusr, $server, $pass)
    bcp ($dbname + ".." + $tbname) in ($indir + "\\" + $basename + "_" + $num + ".csv") -o ($logdir + "\\" +
    $tbname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($fmtfile) -U $sqlusr -S $server -P $pass -b
    2500 -t "," -r \n
}

# Background processing of all partitions
for ($i=1; $i -le $numofparts; $i++)
{
    Write-Output "Submit loading trip and fare partitions # $i"
    if ($sqlauth -eq 0) {
        # Use Windows authentication
        Start-Job -ScriptBlock $ScriptBlock1 -Arg ($dbname, $tbname, $basename, $fmtfile, $indir, $logdir, $i)
    }
    else {
        # Use SQL authentication
        Start-Job -ScriptBlock $ScriptBlock2 -Arg ($dbname, $tbname, $basename, $fmtfile, $indir, $logdir, $i,
        $sqlusr, $server, $pass)
    }
}

Get-Job

# Optional - Wait till all jobs complete and report date and time
$date
While (Get-Job -State "Running") { Start-Sleep 10 }
$date

```

Create indexes to optimize joins and query performance

- If you extract data for modeling from multiple tables, create indexes on the join keys to improve the join performance.
- [Create indexes](#) (clustered or non-clustered) targeting the same filegroup for each partition, for example:

```
CREATE CLUSTERED INDEX <table_idx> ON <table_name>( [include index columns here] )
    ON <TablePScheme>(<partition>field)

-- or,

CREATE INDEX <table_idx> ON <table_name>( [include index columns here] )
    ON <TablePScheme>(<partition>field)
```

NOTE

You may choose to create the indexes before bulk importing the data. Index creation before bulk importing slows down the data loading.

Advanced Analytics Process and Technology in Action Example

For an end-to-end walkthrough example using the Team Data Science Process with a public dataset, see [Team Data Science Process in Action: using SQL Server](#).

Move data from a SQL Server database to SQL Database with Azure Data Factory

3/10/2022 • 8 minutes to read • [Edit Online](#)

This article shows how to move data from a SQL Server database to Azure SQL Database via Azure Blob Storage using the Azure Data Factory (ADF): this method is a supported legacy approach that has the advantages of a replicated staging copy, though [we suggest to look at our data migration page for the latest options](#).

For a table that summarizes various options for moving data to an Azure SQL Database, see [Move data to an Azure SQL Database for Azure Machine Learning](#).

Introduction: What is ADF and when should it be used to migrate data?

Azure Data Factory is a fully managed cloud-based data integration service that orchestrates and automates the movement and transformation of data. The key concept in the ADF model is pipeline. A pipeline is a logical grouping of Activities, each of which defines the actions to perform on the data contained in Datasets. Linked services are used to define the information needed for Data Factory to connect to the data resources.

With ADF, existing data processing services can be composed into data pipelines that are highly available and managed in the cloud. These data pipelines can be scheduled to ingest, prepare, transform, analyze, and publish data, and ADF manages and orchestrates the complex data and processing dependencies. Solutions can be quickly built and deployed in the cloud, connecting a growing number of on-premises and cloud data sources.

Consider using ADF:

- when data needs to be continually migrated in a hybrid scenario that accesses both on-premises and cloud resources
- when the data needs transformations or have business logic added to it when being migrated.

ADF allows for the scheduling and monitoring of jobs using simple JSON scripts that manage the movement of data on a periodic basis. ADF also has other capabilities such as support for complex operations. For more information on ADF, see the documentation at [Azure Data Factory \(ADF\)](#).

The Scenario

We set up an ADF pipeline that composes two data migration activities. Together they move data on a daily basis between a SQL Server database and Azure SQL Database. The two activities are:

- Copy data from a SQL Server database to an Azure Blob Storage account
- Copy data from the Azure Blob Storage account to Azure SQL Database.

NOTE

The steps shown here have been adapted from the more detailed tutorial provided by the ADF team: [Copy data from a SQL Server database to Azure Blob storage](#) References to the relevant sections of that topic are provided when appropriate.

Prerequisites

This tutorial assumes you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).
- An **Azure storage account**. You use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you need to obtain the account key used to access the storage. See [Manage storage account access keys](#).
- Access to an **Azure SQL Database**. If you must set up an Azure SQL Database, the topic [Getting Started with Microsoft Azure SQL Database](#) provides information on how to provision a new instance of an Azure SQL Database.
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

NOTE

This procedure uses the [Azure portal](#).

Upload the data to your SQL Server instance

We use the [NYC Taxi dataset](#) to demonstrate the migration process. The NYC Taxi dataset is available, as noted in that post, on Azure blob storage [NYC Taxi Data](#). The data has two files, the trip_data.csv file, which contains trip details, and the trip_fare.csv file, which contains details of the fare paid for each trip. A sample and description of these files are provided in [NYC Taxi Trips Dataset Description](#).

You can either adapt the procedure provided here to a set of your own data or follow the steps as described by using the NYC Taxi dataset. To upload the NYC Taxi dataset into your SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server database](#).

Create an Azure Data Factory

The instructions for creating a new Azure Data Factory and a resource group in the [Azure portal](#) are provided [Create an Azure Data Factory](#). Name the new ADF instance *adfdsp* and name the resource group created *adfdsp*.

Install and configure Azure Data Factory Integration Runtime

The Integration Runtime is a customer-managed data integration infrastructure used by Azure Data Factory to provide data integration capabilities across different network environments. This runtime was formerly called "Data Management Gateway".

To set up, [follow the instructions for creating a pipeline](#)

Create linked services to connect to the data resources

A linked service defines the information needed for Azure Data Factory to connect to a data resource. We have three resources in this scenario for which linked services are needed:

1. On-premises SQL Server
2. Azure Blob Storage
3. Azure SQL Database

The step-by-step procedure for creating linked services is provided in [Create linked services](#).

Define and create tables to specify how to access the datasets

Create tables that specify the structure, location, and availability of the datasets with the following script-based procedures. JSON files are used to define the tables. For more information on the structure of these files, see [Datasets](#).

NOTE

You should execute the `Add-AzureAccount` cmdlet before executing the `New-AzureDataFactoryTable` cmdlet to confirm that the right Azure subscription is selected for the command execution. For documentation of this cmdlet, see [Add-AzureAccount](#).

The JSON-based definitions in the tables use the following names:

- the **table name** in the SQL Server is *nyctaxi_data*
- the **container name** in the Azure Blob Storage account is *containernamespace*

Three table definitions are needed for this ADF pipeline:

1. [SQL on-premises Table](#)
2. [Blob Table](#)
3. [SQL Azure Table](#)

NOTE

These procedures use Azure PowerShell to define and create the ADF activities. But these tasks can also be accomplished using the Azure portal. For details, see [Create datasets](#).

SQL on-premises Table

The table definition for the SQL Server is specified in the following JSON file:

```
{  
    "name": "OnPremSQLTable",  
    "properties":  
    {  
        "location":  
        {  
            "type": "OnPremisesSqlServerTableLocation",  
            "tableName": "nyctaxi_data",  
            "linkedServiceName": "adfonpremsql"  
        },  
        "availability":  
        {  
            "frequency": "Day",  
            "interval": 1,  
            "waitOnExternal":  
            {  
                "retryInterval": "00:01:00",  
                "retryTimeout": "00:10:00",  
                "maximumRetry": 3  
            }  
        }  
    }  
}
```

The column names were not included here. You can subselect on the column names by including them here (for details check the [ADF documentation](#) topic).

Copy the JSON definition of the table into a file called *onpremtabledef.json* file and save it to a known location (here assumed to be *C:\temp\onpremtabledef.json*). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp -File C:\temp\onpremtabledef.json
```

Blob Table

Definition for the table for the output blob location is in the following (this maps the ingested data from on-premises to Azure blob):

```
{
  "name": "OutputBlobTable",
  "properties":
  {
    "location":
    {
      "type": "AzureBlobLocation",
      "folderPath": "containername",
      "format":
      {
        "type": "TextFormat",
        "columnDelimiter": "\t"
      },
      "linkedServiceName": "adfds"
    },
    "availability":
    {
      "frequency": "Day",
      "interval": 1
    }
  }
}
```

Copy the JSON definition of the table into a file called *bloboutputtabledef.json* file and save it to a known location (here assumed to be *C:\temp\bloboutputtabledef.json*). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName adfdsprg -DataFactoryName adfdsp -File C:\temp\bloboutputtabledef.json
```

SQL Azure Table

Definition for the table for the SQL Azure output is in the following (this schema maps the data coming from the blob):

```
{
  "name": "OutputSQLAzureTable",
  "properties":
  {
    "structure":
    [
      { "name": "column1", "type": "String"},
      { "name": "column2", "type": "String"}
    ],
    "location":
    {
      "type": "AzureSqlTableLocation",
      "tableName": "your_db_name",
      "linkedServiceName": "adfdssqlazure_linked_servicename"
    },
    "availability":
    {
      "frequency": "Day",
      "interval": 1
    }
  }
}
```

Copy the JSON definition of the table into a file called *AzureSqlTable.json* file and save it to a known location (here assumed to be *C:\temp\AzureSqlTable.json*). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName adfdsp -DataFactoryName adfdsp -File
C:\temp\AzureSqlTable.json
```

Define and create the pipeline

Specify the activities that belong to the pipeline and create the pipeline with the following script-based procedures. A JSON file is used to define the pipeline properties.

- The script assumes that the **pipeline name** is *AMLDSPProcessPipeline*.
- Also note that we set the periodicity of the pipeline to be executed on daily basis and use the default execution time for the job (12 am UTC).

NOTE

The following procedures use Azure PowerShell to define and create the ADF pipeline. But this task can also be accomplished using the Azure portal. For details, see [Create pipeline](#).

Using the table definitions provided previously, the pipeline definition for the ADF is specified as follows:

```
{
  "name": "AMLDSPProcessPipeline",
  "properties":
  {
    "description" : "This pipeline has two activities: the first one copies data from SQL Server to Azure Blob, and the second one copies from Azure Blob to Azure Database Table",
    "activities":
    [
      {
        "name": "CopyFromSQLtoBlob",
        "description": "Copy data from SQL Server to blob",
        "type": "CopyActivity",
        "inputs": [ {"name": "OnPremSQLTable"} ],
        "outputs": [ {"name": "OutputBlobTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "SqlSource",
            "sqlReaderQuery": "select * from nyctaxi_data"
          },
          "sink":
          {
            "type": "BlobSink"
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 0,
          "timeout": "01:00:00"
        }
      },
      {
        "name": "CopyFromBlobtoSQLAzure",
        "description": "Push data to Sql Azure",
        "type": "CopyActivity",
        "inputs": [ {"name": "OutputBlobTable"} ],
        "outputs": [ {"name": "OutputSQLAzureTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "BlobSource"
          },
          "sink":
          {
            "type": "SqlSink",
            "WriteBatchTimeout": "00:5:00",
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 2,
          "timeout": "02:00:00"
        }
      }
    ]
  }
}
```

Copy this JSON definition of the pipeline into a file called *pipelinedef.json* file and save it to a known location (here assumed to be *C:\temp\pipelinedef.json*). Create the pipeline in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryPipeline -ResourceGroupName adfdsprg -DataFactoryName adfdsp -File C:\temp\pipelinedef.json
```

Start the Pipeline

The pipeline can now be run using the following command:

```
Set-AzureDataFactoryPipelineActivePeriod -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp -StartTime startdateZ -EndTime enddateZ -Name AMLDSPProcessPipeline
```

The *startdate* and *enddate* parameter values need to be replaced with the actual dates between which you want the pipeline to run.

Once the pipeline executes, you should be able to see the data show up in the container selected for the blob, one file per day.

We have not leveraged the functionality provided by ADF to pipe data incrementally. For more information on how to do this and other capabilities provided by ADF, see the [ADF documentation](#).

Prepare data for enhanced machine learning

3/10/2022 • 6 minutes to read • [Edit Online](#)

Pre-processing and cleaning data are important tasks that must be conducted before a dataset can be used for model training. Raw data is often noisy and unreliable, and may be missing values. Using such data for modeling can produce misleading results. These tasks are part of the Team Data Science Process (TDSP) and typically follow an initial exploration of a dataset used to discover and plan the pre-processing required. For more detailed instructions on the TDSP process, see the steps outlined in the [Team Data Science Process](#).

Pre-processing and cleaning tasks, like the data exploration task, can be carried out in a wide variety of environments, such as SQL or Hive or Azure Machine Learning Studio (classic), and with various tools and languages, such as R or Python, depending where your data is stored and how it is formatted. Since TDSP is iterative in nature, these tasks can take place at various steps in the workflow of the process.

This article introduces various data processing concepts and tasks that can be undertaken either before or after ingesting data into Azure Machine Learning Studio (classic).

For an example of data exploration and pre-processing done inside Azure Machine Learning Studio (classic), see the [Pre-processing data](#) video.

Why pre-process and clean data?

Real world data is gathered from various sources and processes and it may contain irregularities or corrupt data compromising the quality of the dataset. The typical data quality issues that arise are:

- **Incomplete:** Data lacks attributes or containing missing values.
- **Noisy:** Data contains erroneous records or outliers.
- **Inconsistent:** Data contains conflicting records or discrepancies.

Quality data is a prerequisite for quality predictive models. To avoid "garbage in, garbage out" and improve data quality and therefore model performance, it is imperative to conduct a data health screen to spot data issues early and decide on the corresponding data processing and cleaning steps.

What are some typical data health screens that are employed?

We can check the general quality of data by checking:

- The number of **records**.
- The number of **attributes** (or **features**).
- The attribute **data types** (nominal, ordinal, or continuous).
- The number of **missing values**.
- **Well-formed** data.
 - If the data is in TSV or CSV, check that the column separators and line separators always correctly separate columns and lines.
 - If the data is in HTML or XML format, check whether the data is well formed based on their respective standards.
 - Parsing may also be necessary in order to extract structured information from semi-structured or unstructured data.
- **Inconsistent data records.** Check the range of values are allowed. For example, if the data contains student GPA (grade point average), check if the GPA is in the designated range, say 0~4.

When you find issues with data, **processing steps** are necessary, which often involves cleaning missing values, data normalization, discretization, text processing to remove and/or replace embedded characters that may affect data alignment, mixed data types in common fields, and others.

Azure Machine Learning consumes well-formed tabular data. If the data is already in tabular form, data pre-processing can be performed directly with Azure Machine Learning Studio (classic) in the Machine Learning. If data is not in tabular form, say it is in XML, parsing may be required in order to convert the data to tabular form.

What are some of the major tasks in data pre-processing?

- **Data cleaning:** Fill in missing values, detect, and remove noisy data and outliers.
- **Data transformation:** Normalize data to reduce dimensions and noise.
- **Data reduction:** Sample data records or attributes for easier data handling.
- **Data discretization:** Convert continuous attributes to categorical attributes for ease of use with certain machine learning methods.
- **Text cleaning:** remove embedded characters that may cause data misalignment, for example, embedded tabs in a tab-separated data file, embedded new lines that may break records, for example.

The sections below detail some of these data processing steps.

How to deal with missing values?

To deal with missing values, it is best to first identify the reason for the missing values to better handle the problem. Typical missing value handling methods are:

- **Deletion:** Remove records with missing values
- **Dummy substitution:** Replace missing values with a dummy value: e.g, *unknown* for categorical or 0 for numerical values.
- **Mean substitution:** If the missing data is numerical, replace the missing values with the mean.
- **Frequent substitution:** If the missing data is categorical, replace the missing values with the most frequent item
- **Regression substitution:** Use a regression method to replace missing values with regressed values.

How to normalize data?

Data normalization rescales numerical values to a specified range. Popular data normalization methods include:

- **Min-Max Normalization:** Linearly transform the data to a range, say between 0 and 1, where the min value is scaled to 0 and max value to 1.
- **Z-score Normalization:** Scale data based on mean and standard deviation: divide the difference between the data and the mean by the standard deviation.
- **Decimal scaling:** Scale the data by moving the decimal point of the attribute value.

How to discretize data?

Data can be discretized by converting continuous values to nominal attributes or intervals. Some ways of doing this are:

- **Equal-Width Binning:** Divide the range of all possible values of an attribute into N groups of the same size, and assign the values that fall in a bin with the bin number.
- **Equal-Height Binning:** Divide the range of all possible values of an attribute into N groups, each containing the same number of instances, then assign the values that fall in a bin with the bin number.

How to reduce data?

There are various methods to reduce data size for easier data handling. Depending on data size and the domain, the following methods can be applied:

- **Record Sampling:** Sample the data records and only choose the representative subset from the data.
- **Attribute Sampling:** Select only a subset of the most important attributes from the data.
- **Aggregation:** Divide the data into groups and store the numbers for each group. For example, the daily revenue numbers of a restaurant chain over the past 20 years can be aggregated to monthly revenue to reduce the size of the data.

How to clean text data?

Text fields in tabular data may include characters that affect columns alignment and/or record boundaries. For example, embedded tabs in a tab-separated file cause column misalignment, and embedded new line characters break record lines. Improper text encoding handling while writing or reading text leads to information loss, inadvertent introduction of unreadable characters (like nulls), and may also affect text parsing. Careful parsing and editing may be required in order to clean text fields for proper alignment and/or to extract structured data from unstructured or semi-structured text data.

Data exploration offers an early view into the data. A number of data issues can be uncovered during this step and corresponding methods can be applied to address those issues. It is important to ask questions such as what is the source of the issue and how the issue may have been introduced. This process also helps you decide on the data processing steps that need to be taken to resolve them. Identifying the final use cases and personas can also be used to prioritize the data processing effort.

References

Data Mining: Concepts and Techniques, Third Edition, Morgan Kaufmann, 2011, Jiawei Han, Micheline Kamber, and Jian Pei

Explore data in the Team Data Science Process

3/10/2022 • 2 minutes to read • [Edit Online](#)

Exploring data is a step in the [Team Data Science Process](#).

The following articles describe how to explore data in three different storage environments that are typically used in the Data Science Process:

- Explore [Azure blob container](#) data using the [Pandas](#) Python package.
- Explore [SQL Server](#) data by using SQL and by using a programming language like Python.
- Explore [Hive table](#) data using Hive queries.

The [Azure Machine Learning Resources](#) provide documentation and videos on getting started with Azure Machine Learning.

Explore data in Azure Blob storage with the pandas Python package

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article covers how to explore data that is stored in Azure blob container using the [pandas](#) Python package.

This task is a step in the [Team Data Science Process](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Stored your data in an Azure Blob storage account. If you need instructions, see [Moving data to and from Azure Storage](#)

Load the data into a pandas DataFrame

To explore and manipulate a dataset, it must first be downloaded from the blob source to a local file, which can then be loaded in a pandas DataFrame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following Python code sample using Blob service. Replace the variable in the following code with your specific values:

```
from azure.storage.blob import BlobServiceClient
import pandas as pd

STORAGEACCOUNTURL= <storage_account_url>
STORAGEACCOUNTKEY= <storage_account_key>
LOCALFILENAME= <local_file_name>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

#download from blob
t1=time.time()
blob_service_client_instance = BlobServiceClient(account_url=STORAGEACCOUNTURL,
credential=STORAGEACCOUNTKEY)
blob_client_instance = blob_service_client_instance.get_blob_client(CONTAINERNAME, BLOBNAME,
snapshot=None)
with open(LOCALFILENAME, "wb") as my_blob:
    blob_data = blob_client_instance.download_blob()
    blob_data.readinto(my_blob)
t2=time.time()
print("It takes %s seconds to download "+BLOBNAME) % (t2 - t1)
```

2. Read the data into a pandas DataFrame from the downloaded file.

```
# LOCALFILE is the file path
dataframe_blobdata = pd.read_csv(LOCALFILENAME)
```

If you need more general information on reading from an Azure Storage Blob, look at our documentation [Azure Storage Blobs client library for Python](#).

Now you are ready to explore the data and generate features on this dataset.

Examples of data exploration using pandas

Here are a few examples of ways to explore data using pandas:

1. Inspect the **number of rows and columns**

```
print('the size of the data is: %d rows and %d columns' % dataframe_blobdata.shape)
```

2. Inspect the first or last few **rows** in the following dataset:

```
dataframe_blobdata.head(10)  
dataframe_blobdata.tail(10)
```

3. Check the **data type** each column was imported as using the following sample code

```
for col in dataframe_blobdata.columns:  
    print(dataframe_blobdata[col].name, ':', dataframe_blobdata[col].dtype)
```

4. Check the **basic stats** for the columns in the data set as follows

```
dataframe_blobdata.describe()
```

5. Look at the number of entries for each column value as follows

```
dataframe_blobdata['<column_name>'].value_counts()
```

6. Count missing values versus the actual number of entries in each column using the following sample code

```
miss_num = dataframe_blobdata.shape[0] - dataframe_blobdata.count()  
print(miss_num)
```

7. If you have **missing values** for a specific column in the data, you can drop them as follows:

```
dataframe_blobdata_noNA = dataframe_blobdata.dropna()  
dataframe_blobdata_noNA.shape
```

Another way to replace missing values is with the mode function:

```
dataframe_blobdata_mode = dataframe_blobdata.fillna(  
    {'<column_name>': dataframe_blobdata['<column_name>'].mode()[0]})
```

8. Create a **histogram** plot using variable number of bins to plot the distribution of a variable

```
dataframe_blobdata['<column_name>'].value_counts().plot(kind='bar')  
np.log(dataframe_blobdata['<column_name>']+1).hist(bins=50)
```

9. Look at **correlations** between variables using a scatterplot or using the built-in correlation function

```
# relationship between column_a and column_b using scatter plot  
plt.scatter(dataframe_blobdata['<column_a>'], dataframe_blobdata['<column_b>'])  
  
# correlation between column_a and column_b  
dataframe_blobdata[['<column_a>', '<column_b>']].corr()
```

Explore data in a SQL Server virtual machine on Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article covers how to explore data that is stored in a SQL Server VM on Azure. Use SQL or Python to examine the data.

This task is a step in the [Team Data Science Process](#).

NOTE

The sample SQL statements in this document assume that data is in SQL Server. If it isn't, refer to the cloud data science process map to learn how to move your data to SQL Server.

Explore SQL data with SQL scripts

Here are a few sample SQL scripts that can be used to explore data stores in SQL Server.

1. Get the count of observations per day

```
SELECT CONVERT(date, <date_columnname>) as date, count(*) as c from <tablename> group by CONVERT(date, <date_columnname>)
```

2. Get the levels in a categorical column

```
select distinct <column_name> from <databasename>
```

3. Get the number of levels in combination of two categorical columns

```
select <column_a>, <column_b>, count(*) from <tablename> group by <column_a>, <column_b>
```

4. Get the distribution for numerical columns

```
select <column_name>, count(*) from <tablename> group by <column_name>
```

NOTE

For a practical example, you can use the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

Explore SQL data with Python

Using Python to explore data and generate features when the data is in SQL Server is similar to processing data in Azure blob using Python, as documented in [Process Azure Blob data in your data science environment](#). Load the data from the database into a pandas DataFrame and then can be processed further. We document the process of connecting to the database and loading the data into the DataFrame in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replace servername, dbname, username, and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The following code reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql('''select <columnname1>, <columnname2>... from <tablename>''', conn)
```

Now you can work with the Pandas DataFrame as covered in the topic [Process Azure Blob data in your data science environment](#).

The Team Data Science Process in action example

For an end-to-end walkthrough example of the Cortana Analytics Process using a public dataset, see [The Team Data Science Process in action: using SQL Server](#).

Explore data in Hive tables with Hive queries

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article provides sample Hive scripts that are used to explore data in Hive tables in an HDInsight Hadoop cluster.

This task is a step in the [Team Data Science Process](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Customize Azure HDInsight Hadoop Clusters for Advanced Analytics](#).
- The data has been uploaded to Hive tables in Azure HDInsight Hadoop clusters. If it has not, follow the instructions in [Create and load data to Hive tables](#) to upload data to Hive tables first.
- Enabled remote access to the cluster. If you need instructions, see [Access the Head Node of Hadoop Cluster](#).
- If you need instructions on how to submit Hive queries, see [How to Submit Hive Queries](#)

Example Hive query scripts for data exploration

1. Get the count of observations per partition

```
SELECT <partitionfieldname>, count(*) from <databasename>.<tablename> group by <partitionfieldname>;
```

2. Get the count of observations per day

```
SELECT to_date(<date_columnname>), count(*) from <databasename>.<tablename> group by to_date(<date_columnname>);
```

3. Get the levels in a categorical column

```
SELECT distinct <column_name> from <databasename>.<tablename>
```

4. Get the number of levels in combination of two categorical columns

```
SELECT <column_a>, <column_b>, count(*) from <databasename>.<tablename> group by <column_a>, <column_b>;
```

5. Get the distribution for numerical columns

```
SELECT <column_name>, count(*) from <databasename>.<tablename> group by <column_name>
```

6. Extract records from joining two tables

```

SELECT
    a.<common_columnname1> as <new_name1>,
    a.<common_columnname2> as <new_name2>,
    a.<a_column_name1> as <new_name3>,
    a.<a_column_name2> as <new_name4>,
    b.<b_column_name1> as <new_name5>,
    b.<b_column_name2> as <new_name6>
FROM
(
    SELECT <common_columnname1>,
        <common_columnname2>,
        <a_column_name1>,
        <a_column_name2>,
    FROM <databasename>.<tablename1>
) a
join
(
    SELECT <common_columnname1>,
        <common_columnname2>,
        <b_column_name1>,
        <b_column_name2>,
    FROM <databasename>.<tablename2>
) b
ON a.<common_columnname1>=b.<common_columnname1> and a.<common_columnname2>=b.
<common_columnname2>

```

Additional query scripts for taxi trip data scenarios

Examples of queries that are specific to NYC Taxi Trip Data scenarios are also provided in [GitHub repository](#).

These queries already have data schema specified and are ready to be submitted to run. The NYC Taxi Trip data is available through [Azure Open Datasets](#) or from the source [TLC Trip Record Data](#).

Sample data in Azure blob containers, SQL Server, and Hive tables

3/10/2022 • 2 minutes to read • [Edit Online](#)

The following articles describe how to sample data that is stored in one of three different Azure locations:

- [Azure blob container data](#) is sampled by downloading it programmatically and then sampling it with sample Python code.
- [SQL Server data](#) is sampled using both SQL and the Python Programming Language.
- [Hive table data](#) is sampled using Hive queries.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Why sample data?

If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. Downsizing may facilitate data understanding, exploration, and feature engineering. This sampling role in the Cortana Analytics Process is to enable fast prototyping of the data processing functions and machine learning models.

Sample data in Azure Blob storage

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article covers sampling data stored in Azure Blob storage by downloading it programmatically and then sampling it using procedures written in Python.

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. Sampling facilitates data understanding, exploration, and feature engineering. Its role in the Cortana Analytics Process is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Download and down-sample data

1. Download the data from Azure Blob storage using the Blob service from the following sample Python code:

```
from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME= <storage_account_name>
STORAGEACCOUNTKEY= <storage_account_key>
LOCALFILENAME= <local_file_name>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

#download from blob
t1=time.time()
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)
t2=time.time()
print(("It takes %s seconds to download "+blobname) % (t2 - t1))
```

2. Read data into a Pandas data-frame from the file downloaded above.

```
import pandas as pd

#directly ready from file on disk
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

3. Down-sample the data using the `numpy`'s `random.choice` as follows:

```
# A 1 percent sample
sample_ratio = 0.01
sample_size = np.round(dataframe_blobdata.shape[0] * sample_ratio)
sample_rows = np.random.choice(dataframe_blobdata.index.values, sample_size)
dataframe_blobdata_sample = dataframe_blobdata.ix[sample_rows]
```

Now you can work with the above data frame with the one Percent sample for further exploration and feature generation.

Upload data and read it into Azure Machine Learning

You can use the following sample code to down-sample the data and use it directly in Azure Machine Learning:

1. Write the data frame to a local file

```
dataframe.to_csv(os.path.join(os.getcwd(),LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload the local file to an Azure blob using the following sample code:

```
from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME= <storage_account_name>
LOCALFILENAME= <local_file_name>
STORAGEACCOUNTKEY= <storage_account_key>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

try:

    #perform upload
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading to the blob:"+ BLOBNAME)
```

3. Make a datastore in Azure Machine Learning which points to the Azure Blob Storage. [This link describes the concept of datastores and how to subsequently make a dataset for use with Azure Machine Learning.](#)

Sample data in SQL Server on Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article shows how to sample data stored in SQL Server on Azure using either SQL or the Python programming language. It also shows how to move sampled data into Azure Machine Learning by saving it to a file, uploading it to an Azure blob, and then reading it into Azure Machine Learning Studio.

The Python sampling uses the [pyodbc](#) ODBC library to connect to SQL Server on Azure and the [Pandas](#) library to do the sampling.

NOTE

The sample SQL code in this document assumes that the data is in a SQL Server on Azure. If it is not, refer to [Move data to SQL Server on Azure](#) article for instructions on how to move your data to SQL Server on Azure.

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. Sampling facilitates data understanding, exploration, and feature engineering. Its role in the [Team Data Science Process \(TDSP\)](#) is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Using SQL

This section describes several methods using SQL to perform simple random sampling against the data in the database. Choose a method based on your data size and its distribution.

The following two items show how to use `newid` in SQL Server to perform the sampling. The method you choose depends on how random you want the sample to be (`pk_id` in the following sample code is assumed to be an autogenerated primary key).

1. Less strict random sample

```
select * from <table_name> where <primary_key> in  
(select top 10 percent <primary_key> from <table_name> order by newid())
```

2. More random sample

```
SELECT * FROM <table_name>  
WHERE 0.1 >= CAST(CHECKSUM(NEWID(), <primary_key>) & 0x7fffffff AS float)/ CAST (0x7fffffff AS int)
```

`Tablesample` can be used for sampling the data as well. This option may be a better approach if your data size is large (assuming that data on different pages is not correlated) and for the query to complete in a reasonable time.

```
SELECT *  
FROM <table_name>  
TABLESAMPLE (10 PERCENT)
```

NOTE

You can explore and generate features from this sampled data by storing it in a new table

Connecting to Azure Machine Learning

You may directly use the sample queries above in Azure Machine Learning code (perhaps a notebook, or code inserted into Designer). [See this link for more details about how to connect to storage with an Azure Machine Learning datastore.](#)

Using the Python programming language

This section demonstrates using the [pyodbc library](#) to establish an ODBC connect to a SQL server database in Python. The database connection string is as follows: (replaceservername, dbname, username, and password with your configuration):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas](#) library in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The following code reads a 0.1% sample of the data from a table in Azure SQL Database into a Pandas data:

```
import pandas as pd

# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql('''select column1, column2... from <table_name> tablesample (0.1 percent)'', conn)
```

You can now work with the sampled data in the Pandas data frame.

Connecting to Azure Machine Learning

You can use the following sample code to save the down-sampled data to a file and upload it to an Azure blob. The data in the blob can be directly read into an Azure Machine Learning Experiment using the [Import Data](#) module. The steps are as follows:

1. Write the pandas data frame to a local file

```
dataframe.to_csv(os.path.join(os.getcwd(),LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload local file to Azure blob

```
from azure.storage import BlobService
import tables

STORAGEACCOUNTNAME= <storage_account_name>
LOCALFILENAME= <local_file_name>
STORAGEACCOUNTKEY= <storage_account_key>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working
directory

try:
    #perform upload
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading blob:"+BLOBNAME)
```

3. This guide provides an overview of the next step to access data in Azure Machine Learning through datastores and datasets.

The Team Data Science Process in Action example

To walk through an example of the Team Data Science Process a using a public dataset, see [Team Data Science Process in Action: using SQL Server](#).

Sample data in Azure HDInsight Hive tables

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article describes how to down-sample data stored in Azure HDInsight Hive tables using Hive queries to reduce it to a size more manageable for analysis. It covers three popularly used sampling methods:

- Uniform random sampling
- Random sampling by groups
- Stratified sampling

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. Down-sampling facilitates data understanding, exploration, and feature engineering. Its role in the Team Data Science Process is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

How to submit Hive queries

Hive queries can be submitted from the Hadoop Command-Line console on the head node of the Hadoop cluster. Log into the head node of the Hadoop cluster, open the Hadoop Command-Line console, and submit the Hive queries from there. For instructions on submitting Hive queries in the Hadoop Command-Line console, see [How to Submit Hive Queries](#).

Uniform random sampling

Uniform random sampling means that each row in the data set has an equal chance of being sampled. It can be implemented by adding an extra field rand() to the data set in the inner "select" query, and in the outer "select" query that condition on that random field.

Here is an example query:

```
SET sampleRate=<sample rate, 0-1>;
select
    field1, field2, ..., fieldN
from
    (
    select
        field1, field2, ..., fieldN, rand() as samplekey
    from <hive table name>
    )a
where samplekey='${hiveconf:sampleRate}'
```

Here, `<sample rate, 0-1>` specifies the proportion of records that the users want to sample.

Random sampling by groups

When sampling categorical data, you may want to either include or exclude all of the instances for some value of the categorical variable. This sort of sampling is called "sampling by group". For example, if you have a categorical variable "*State*", which has values such as NY, MA, CA, NJ, and PA, you want records from each state to be together, whether they are sampled or not.

Here is an example query that samples by group:

```

SET sampleRate=<sample rate, 0-1>;
select
    b.field1, b.field2, ..., b.catfield, ..., b.fieldN
from
(
    select
        field1, field2, ..., catfield, ..., fieldN
    from <table name>
) b
join
(
    select
        catfield
    from
    (
        select
            catfield, rand() as samplekey
        from <table name>
        group by catfield
    ) a
    where samplekey<='${hiveconf:sampleRate}'
) c
on b.catfield=c.catfield

```

Stratified sampling

Random sampling is stratified with respect to a categorical variable when the samples obtained have categorical values that are present in the same ratio as they were in the parent population. Using the same example as above, suppose your data has the following observations by states: NJ has 100 observations, NY has 60 observations, and WA has 300 observations. If you specify the rate of stratified sampling to be 0.5, then the sample obtained should have approximately 50, 30, and 150 observations of NJ, NY, and WA respectively.

Here is an example query:

```

SET sampleRate=<sample rate, 0-1>;
select
    field1, field2, field3, ..., fieldN, state
from
(
    select
        field1, field2, field3, ..., fieldN, state,
        count(*) over (partition by state) as state_cnt,
        rank() over (partition by state order by rand()) as state_rank
    from <table name>
) a
where state_rank <= state_cnt* '${hiveconf:sampleRate}'

```

For information on more advanced sampling methods that are available in Hive, see [LanguageManual Sampling](#).

Access datasets with Python using the Azure Machine Learning Python client library

3/10/2022 • 8 minutes to read • [Edit Online](#)

The preview of Microsoft Azure Machine Learning Python client library can enable secure access to your Azure Machine Learning datasets from a local Python environment and enables the creation and management of datasets in a workspace.

This topic provides instructions on how to:

- install the Machine Learning Python client library
- access and upload datasets, including instructions on how to get authorization to access Azure Machine Learning datasets from your local Python environment
- access intermediate datasets from experiments
- use the Python client library to enumerate datasets, access metadata, read the contents of a dataset, create new datasets, and update existing datasets

Prerequisites

The Python client library has been tested under the following environments:

- Windows, Mac, and Linux
- Python 2.7 and 3.6+

It has a dependency on the following packages:

- requests
- python-dateutil
- pandas

We recommend using a Python distribution such as [Anaconda](#) or [Canopy](#), which come with Python, IPython and the three packages listed above installed. Although IPython is not strictly required, it is a great environment for manipulating and visualizing data interactively.

How to install the Azure Machine Learning Python client library

Install the Azure Machine Learning Python client library to complete the tasks outlined in this topic. This library is available from the [Python Package Index](#). To install it in your Python environment, run the following command from your local Python environment:

```
pip install azureml
```

Alternatively, you can download and install from the sources on [GitHub](#).

```
python setup.py install
```

If you have git installed on your machine, you can use pip to install directly from the git repository:

```
pip install git+https://github.com/Azure/Azure-MachineLearning-ClientLibrary-Python.git
```

Use code snippets to access datasets

The Python client library gives you programmatic access to your existing datasets from experiments that have been run.

From the Azure Machine Learning Studio (classic) web interface, you can generate code snippets that include all the necessary information to download and deserialize datasets as pandas DataFrame objects on your local machine.

Security for data access

The code snippets provided by Azure Machine Learning Studio (classic) for use with the Python client library includes your workspace ID and authorization token. These provide full access to your workspace and must be protected, like a password.

For security reasons, the code snippet functionality is only available to users that have their role set as **Owner** for the workspace. Your role is displayed in Azure Machine Learning Studio (classic) on the **USERS** page under **Settings**.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio (classic). On the left, there's a sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS, TRAINED MODELS, and SETTINGS. The SETTINGS icon is highlighted. The main area is titled 'settings' and contains a table for 'USERS'. The table has columns for NAME, AUTHORIZATION TOKENS, and USERS. The 'NAME' column lists a user with a blacked-out name. The 'AUTHORIZATION TOKENS' column is empty. The 'USERS' column shows the user's 'ROLE' as 'Owner' and 'STATUS' as 'Active'. There is a small 'p' icon in the top right corner of the table.

If your role is not set as **Owner**, you can either request to be reinvited as an owner, or ask the owner of the workspace to provide you with the code snippet.

To obtain the authorization token, you may choose one of these options:

- Ask for a token from an owner. Owners can access their authorization tokens from the Settings page of their workspace in Azure Machine Learning Studio (classic). Select **Settings** from the left pane and click **AUTHORIZATION TOKENS** to see the primary and secondary tokens. Although either the primary or the secondary authorization tokens can be used in the code snippet, it is recommended that owners only share the secondary authorization tokens.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio (classic). On the left, there's a sidebar with icons for EXPERIMENTS, WEB SERVICES, and SETTINGS. The SETTINGS icon is highlighted with a red oval. The main area is titled 'settings' and contains a table for 'AUTHORIZATION TOKENS'. The table has columns for NAME, PRIMARY AUTHORIZATION TOKEN, and SECONDARY AUTHORIZATION TOKEN. The 'NAME' column is circled in red. The 'PRIMARY AUTHORIZATION TOKEN' and 'SECONDARY AUTHORIZATION TOKEN' columns each have a blacked-out token value and a 'Regenerate' button. A red oval also surrounds the 'Regenerate' button for the secondary token.

- Ask to be promoted to role of owner: a current owner of the workspace needs to first remove you from the workspace then reinvite you to it as an owner.

Once developers have obtained the workspace ID and authorization token, they are able to access the workspace using the code snippet regardless of their role.

Authorization tokens are managed on the **AUTHORIZATION TOKENS** page under **SETTINGS**. You can regenerate them, but this procedure revokes access to the previous token.

Access datasets from a local Python application

1. In Machine Learning Studio (classic), click DATASETS in the navigation bar on the left.
2. Select the dataset you would like to access. You can select any of the datasets from the **MY DATASETS** list or from the **SAMPLES** list.
3. From the bottom toolbar, click **Generate Data Access Code**. If the data is in a format incompatible with the Python client library, this button is disabled.

The screenshot shows the Machine Learning Studio (classic) interface. On the left, there is a sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS (which is selected and highlighted in blue), TRAINED MODELS, and SETTINGS. The main area is titled "datasets". It shows a table with two rows of data:

NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE	CREA... ↴	Ρ
My Data.tsv	ptvsazure	My Test Data	GenericTSV	1/20/2015 12:3...	

At the bottom of the screen, there is a dark footer bar with icons for NEW, DOWNLOAD, DELETE, and GENERATE DATA ACCESS CODE. The "GENERATE DATA ACCESS CODE" icon is highlighted with a red box.

4. Select the code snippet from the window that appears and copy it to your clipboard.

The screenshot shows the "GENERATE DATA ACCESS CODE" dialog box. It contains the following text:

Use this code to access your data

To programmatically access this dataset, copy the code snippet into your favorite development environment. [Learn More](#).

Note: this code includes your workspace access token, which provides full access to your workspace. It should be treated like a password.

CODE SNIPPET

Python

```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]')
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()
```

USE SECONDARY TOKEN

A checkmark icon is located at the bottom right of the dialog.

5. Paste the code into the notebook of your local Python application.

The screenshot shows an IPython Notebook interface with two code cells and their outputs.

In [3]:

```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]')
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()
```

In [4]:

```
frame
```

Out[4]:

	fLength	fWidth	fSize	fConc	fConcl	fAsym	fM3Long	fM3Trans	fAlpha	fDist	Class
0	29.4491	12.7271	2.6637	0.3536	0.1876	-19.4070	-18.1295	7.1258	8.1501	252.1500	g
1	51.5830	10.7969	2.6222	0.5227	0.2733	-64.0583	-30.1280	4.3855	15.0428	269.4810	g
2	36.3558	10.3843	2.8531	0.5309	0.3599	15.4179	51.9328	16.2848	9.1263	208.7935	h
3	37.2577	12.0793	2.4354	0.3560	0.2037	5.1882	-17.8545	6.0370	30.0150	61.1727	h
4	34.8906	15.7072	2.7147	0.3587	0.1938	-8.5682	-12.6514	-14.3676	89.7920	222.4690	h
5	60.4957	17.6753	2.9380	0.1753	0.0894	31.3257	-15.9801	14.4117	15.6390	113.1820	g
6	18.5731	16.2365	2.6758	0.4895	0.3091	1.1158	8.1104	-11.7988	50.9791	224.8780	h
7	21.5929	12.4539	2.3512	0.4900	0.3096	0.1172	-4.3583	2.5525	58.3290	52.0772	g
8	45.5590	9.9957	2.5809	0.3885	0.1955	17.0284	34.9608	-7.8856	14.4173	177.6820	g
9	62.3597	21.6946	3.1739	0.3087	0.2041	-45.3412	52.1023	22.5905	36.9876	274.7409	h

Access intermediate datasets from Machine Learning experiments

After an experiment is run in Machine Learning Studio (classic), it is possible to access the intermediate datasets from the output nodes of modules. Intermediate datasets are data that has been created and used for intermediate steps when a model tool has been run.

Intermediate datasets can be accessed as long as the data format is compatible with the Python client library.

The following formats are supported (constants for these formats are in the `azureml.DataTypeIds` class):

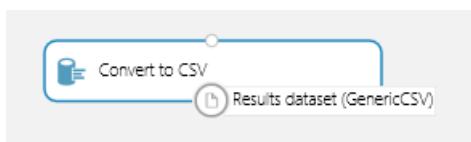
- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

You can determine the format by hovering over a module output node. It is displayed along with the node name, in a tooltip.

Some of the modules, such as the [Split](#) module, output to a format named `Dataset`, which is not supported by the Python client library.



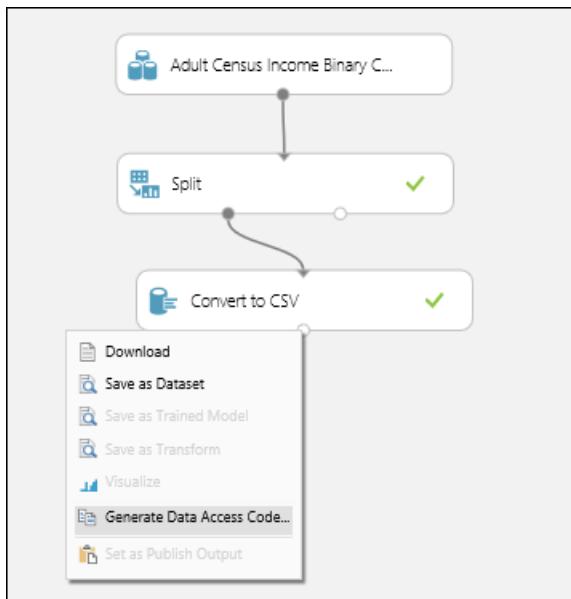
You need to use a conversion module, such as [Convert to CSV](#), to get an output into a supported format.



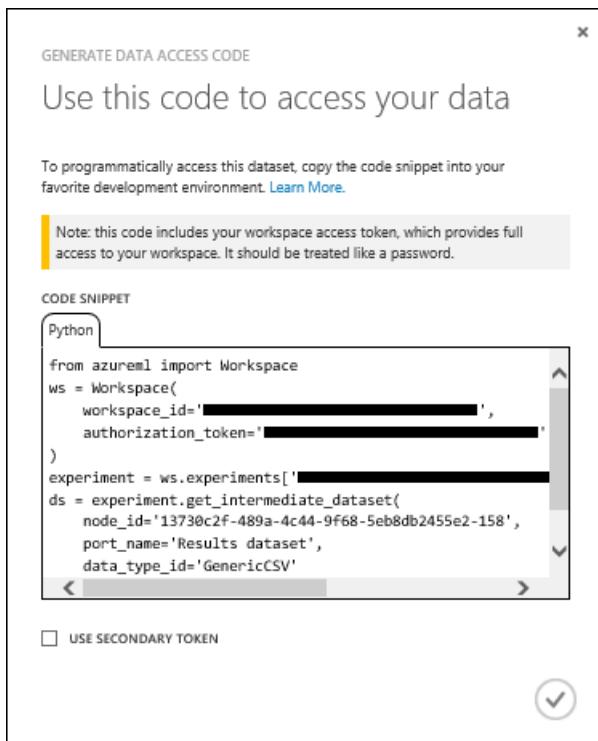
The following steps show an example that creates an experiment, runs it and accesses the intermediate dataset.

1. Create a new experiment.

2. Insert an **Adult Census Income Binary Classification** dataset module.
3. Insert a **Split** module, and connect its input to the dataset module output.
4. Insert a **Convert to CSV** module and connect its input to one of the **Split** module outputs.
5. Save the experiment, run it, and wait for the job to finish.
6. Click the output node on the **Convert to CSV** module.
7. When the context menu appears, select **Generate Data Access Code**.



8. Select the code snippet and copy it to your clipboard from the window that appears.



9. Paste the code in your notebook.

IP[y]: Notebook My Test Notebook Last Checkpoint: Jan 20 12:58 (unsaved changes)

File Edit View Insert Cell Kernel Help

In [6]:

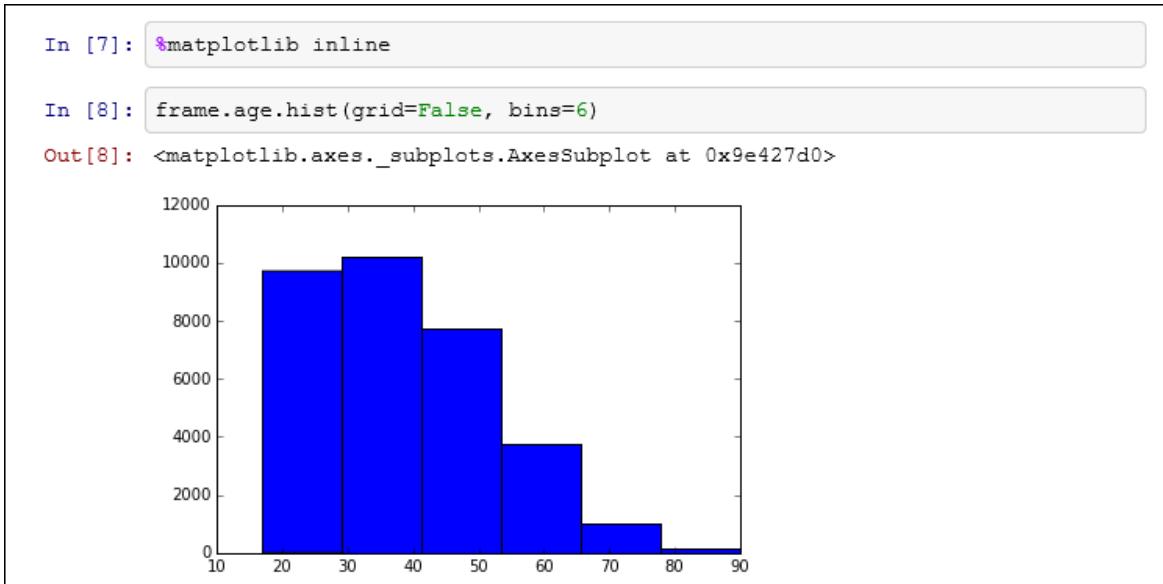
```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]'
)
experiment = ws.experiments['[REDACTED]']
ds = experiment.get_intermediate_dataset(
    node_id='13730c2f-489a-4c44-9f68-5eb8db2455e2-158',
    port_name='Results dataset',
    data_type_id='GenericCSV'
)
frame = ds.to_dataframe()
```

In [7]: frame

Out[7]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	52	Private	225317	5th-6th	3	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0
1	29	Private	154017	HS-grad	9	Never-married	Sales	Not-in-family	White	Female	0	0
2	25	Private	203570	HS-grad	9	Separated	Other-service	Unmarried	Black	Male	0	0

10. You can visualize the data using matplotlib. This displays in a histogram for the age column:



Use the Machine Learning Python client library to access, read, create, and manage datasets

Workspace

The workspace is the entry point for the Python client library. Provide the `Workspace` class with your workspace ID and authorization token to create an instance:

```
ws = Workspace(workspace_id='4c29e1adeba2e5a7cbeb0e4f4adfb4df',
               authorization_token='f4f3ade2c6aefdb1afb043cd8bcf3daf')
```

Enumerate datasets

To enumerate all datasets in a given workspace:

```
for ds in ws.datasets:  
    print(ds.name)
```

To enumerate just the user-created datasets:

```
for ds in ws.user_datasets:  
    print(ds.name)
```

To enumerate just the example datasets:

```
for ds in ws.example_datasets:  
    print(ds.name)
```

You can access a dataset by name (which is case-sensitive):

```
ds = ws.datasets['my dataset name']
```

Or you can access it by index:

```
ds = ws.datasets[0]
```

Metadata

Datasets have metadata, in addition to content. (Intermediate datasets are an exception to this rule and do not have any metadata.)

Some metadata values are assigned by the user at creation time:

- `print(ds.name)`
- `print(ds.description)`
- `print(ds.family_id)`
- `print(ds.data_type_id)`

Others are values assigned by Azure ML:

- `print(ds.id)`
- `print(ds.created_date)`
- `print(ds.size)`

See the `SourceDataset` class for more on the available metadata.

Read contents

The code snippets provided by Machine Learning Studio (classic) automatically download and deserialize the dataset to a pandas DataFrame object. This is done with the `to_dataframe` method:

```
frame = ds.to_dataframe()
```

If you prefer to download the raw data, and perform the deserialization yourself, that is an option. At the moment, this is the only option for formats such as 'ARFF', which the Python client library cannot deserialize.

To read the contents as text:

```
text_data = ds.read_as_text()
```

To read the contents as binary:

```
binary_data = ds.read_as_binary()
```

You can also just open a stream to the contents:

```
with ds.open() as file:  
    binary_data_chunk = file.read(1000)
```

Create a new dataset

The Python client library allows you to upload datasets from your Python program. These datasets are then available for use in your workspace.

If you have your data in a pandas DataFrame, use the following code:

```
from azureml import DataTypeIds  
  
dataset = ws.datasets.add_from_dataframe(  
    dataframe=frame,  
    data_type_id=DataTypeIds.GenericCSV,  
    name='my new dataset',  
    description='my description'  
)
```

If your data is already serialized, you can use:

```
from azureml import DataTypeIds  
  
dataset = ws.datasets.add_from_raw_data(  
    raw_data=raw_data,  
    data_type_id=DataTypeIds.GenericCSV,  
    name='my new dataset',  
    description='my description'  
)
```

The Python client library is able to serialize a pandas DataFrame to the following formats (constants for these are in the `azureml.DataTypeIds` class):

- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

Update an existing dataset

If you try to upload a new dataset with a name that matches an existing dataset, you should get a conflict error.

To update an existing dataset, you first need to get a reference to the existing dataset:

```
dataset = ws.datasets['existing dataset']

print(dataset.data_type_id) # 'GenericCSV'
print(dataset.name)        # 'existing dataset'
print(dataset.description) # 'data up to jan 2015'
```

Then use `update_from_dataframe` to serialize and replace the contents of the dataset on Azure:

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(frame2)

print(dataset.data_type_id) # 'GenericCSV'
print(dataset.name)        # 'existing dataset'
print(dataset.description) # 'data up to jan 2015'
```

If you want to serialize the data to a different format, specify a value for the optional `data_type_id` parameter.

```
from azureml import DataTypeIds

dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    datafram=frame2,
    data_type_id=DataTypeIds.GenericTSV,
)

print(dataset.data_type_id) # 'GenericTSV'
print(dataset.name)        # 'existing dataset'
print(dataset.description) # 'data up to jan 2015'
```

You can optionally set a new description by specifying a value for the `description` parameter.

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    datafram=frame2,
    description='data up to feb 2015',
)

print(dataset.data_type_id) # 'GenericCSV'
print(dataset.name)        # 'existing dataset'
print(dataset.description) # 'data up to feb 2015'
```

You can optionally set a new name by specifying a value for the `name` parameter. From now on, you'll retrieve the dataset using the new name only. The following code updates the data, name, and description.

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    dataframe=frame2,
    name='existing dataset v2',
    description='data up to feb 2015',
)

print(dataset.data_type_id)                      # 'GenericCSV'
print(dataset.name)                            # 'existing dataset v2'
print(dataset.description)                     # 'data up to feb 2015'

print(ws.datasets['existing dataset v2'].name) # 'existing dataset v2'
print(ws.datasets['existing dataset'].name)    # IndexError
```

The `data_type_id`, `name` and `description` parameters are optional and default to their previous value. The `dataframe` parameter is always required.

If your data is already serialized, use `update_from_raw_data` instead of `update_from_dataframe`. If you just pass in `raw_data` instead of `dataframe`, it works in a similar way.

Process Azure blob data with advanced analytics

3/10/2022 • 3 minutes to read • [Edit Online](#)

This document covers exploring data and generating features from data stored in Azure Blob storage.

Load the data into a Pandas data frame

In order to explore and manipulate a dataset, it must be downloaded from the blob source to a local file that can then be loaded in a Pandas data frame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following sample Python code using Blob service. Replace the variable in the code below with your specific values:

```
from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME= <storage_account_name>
STORAGEACCOUNTKEY= <storage_account_key>
LOCALFILENAME= <local_file_name>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

#download from blob
t1=time.time()
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)
t2=time.time()
print("It takes %s seconds to download "+blobname) % (t2 - t1)
```

2. Read the data into a Pandas data-frame from the downloaded file.

```
#LOCALFILE is the file path
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

Now you are ready to explore the data and generate features on this dataset.

Data Exploration

Here are a few examples of ways to explore data using Pandas:

1. Inspect the number of rows and columns:

```
print 'the size of the data is: %d rows and %d columns' % dataframe_blobdata.shape
```

2. Inspect the first or last few rows in the dataset as below:

```
dataframe_blobdata.head(10)

dataframe_blobdata.tail(10)
```

3. Check the data type each column was imported as using the following sample code

```
for col in dataframe_blobdata.columns:  
    print dataframe_blobdata[col].name, ':\t', dataframe_blobdata[col].dtype
```

4. Check the basic stats for the columns in the data set as follows

```
dataframe_blobdata.describe()
```

5. Look at the number of entries for each column value as follows

```
dataframe_blobdata['<column_name>'].value_counts()
```

6. Count missing values versus the actual number of entries in each column using the following sample code

```
miss_num = dataframe_blobdata.shape[0] - dataframe_blobdata.count()  
print miss_num
```

7. If you have missing values for a specific column in the data, you can drop them as follows:

```
dataframe_blobdata_noNA = dataframe_blobdata.dropna()  
dataframe_blobdata_noNA.shape
```

Another way to replace missing values is with the mode function:

```
dataframe_blobdata_mode =  
dataframe_blobdata.fillna({'<column_name>':dataframe_blobdata['<column_name>'].mode()[0]})
```

8. Create a histogram plot using variable number of bins to plot the distribution of a variable:

```
dataframe_blobdata['<column_name>'].value_counts().plot(kind='bar')  
  
np.log(dataframe_blobdata['<column_name>']+1).hist(bins=50)
```

9. Look at correlations between variables using a scatterplot or using the built-in correlation function:

```
#relationship between column_a and column_b using scatter plot  
plt.scatter(dataframe_blobdata['<column_a>'], dataframe_blobdata['<column_b>'])  
  
#correlation between column_a and column_b  
dataframe_blobdata[['<column_a>', '<column_b>']].corr()
```

Feature Generation

We can generate features using Python as follows:

Indicator value-based Feature Generation

Categorical features can be created as follows:

1. Inspect the distribution of the categorical column:

```
dataframe_blobdata['<categorical_column>'].value_counts()
```

2. Generate indicator values for each of the column values:

```
#generate the indicator column
dataframe_blobdata_identity = pd.get_dummies(dataframe_blobdata['<categorical_column>'],
prefix='<categorical_column>_identity')
```

3. Join the indicator column with the original data frame:

```
#Join the dummy variables back to the original data frame
dataframe_blobdata_with_identity = dataframe_blobdata.join(dataframe_blobdata_identity)
```

4. Remove the original variable itself:

```
#Remove the original column rate_code in df1_with_dummy
dataframe_blobdata_with_identity.drop('<categorical_column>', axis=1, inplace=True)
```

Binning Feature Generation

For generating binned features, we proceed as follows:

1. Add a sequence of columns to bin a numeric column:

```
bins = [0, 1, 2, 4, 10, 40]
dataframe_blobdata_bin_id = pd.cut(dataframe_blobdata['<numeric_column>'], bins)
```

2. Convert binning to a sequence of boolean variables

```
dataframe_blobdata_bin_bool = pd.get_dummies(dataframe_blobdata_bin_id, prefix='<numeric_column>')
```

3. Finally, Join the dummy variables back to the original data frame

```
dataframe_blobdata_with_bin_bool = dataframe_blobdata.join(dataframe_blobdata_bin_bool)
```

Writing data back to Azure blob and consuming in Azure Machine Learning

After you have explored the data and created the necessary features, you can upload the data (sampled or featurized) to an Azure blob and consume it in Azure Machine Learning using the following steps: Additional features can be created in the Azure Machine Learning Studio (classic) as well.

1. Write the data frame to local file

```
dataframe.to_csv(os.path.join(os.getcwd(), LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload the data to Azure blob as follows:

```

from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME= <storage_account_name>
LOCALFILENAME= <local_file_name>
STORAGEACCOUNTKEY= <storage_account_key>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working
directory

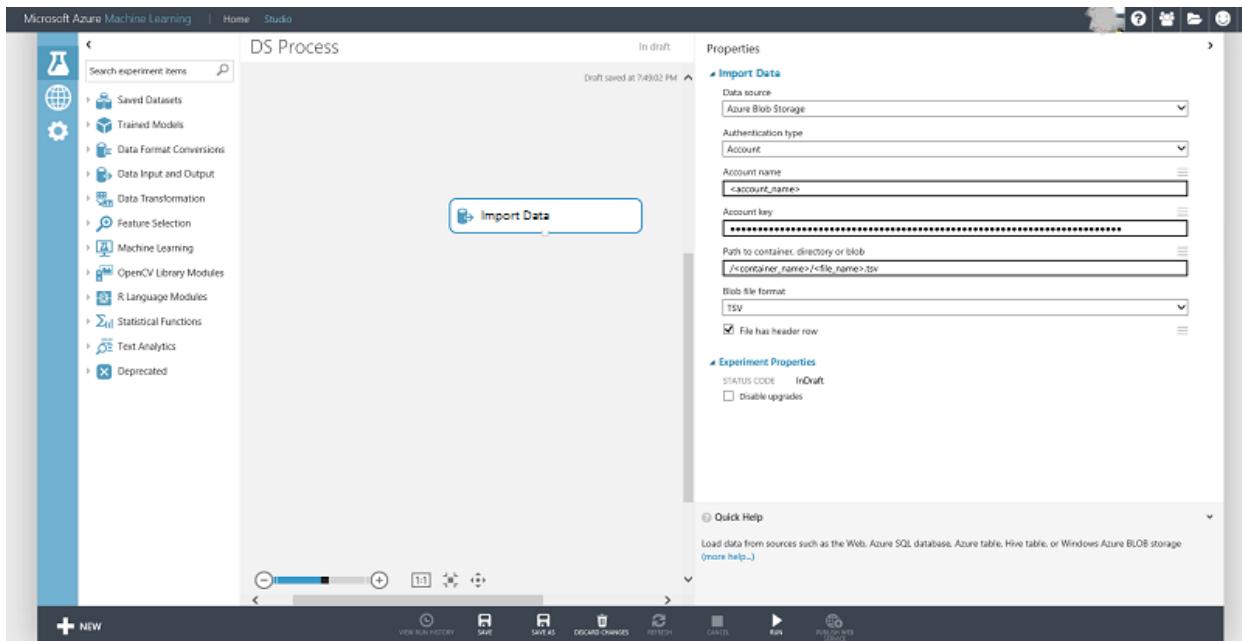
try:

#perform upload
output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading blob:"+BLOBNAME)

```

3. Now the data can be read from the blob using the Azure Machine Learning [Import Data](#) module as shown in the screen below:



Scalable data science with Azure Data Lake

3/10/2022 • 19 minutes to read • [Edit Online](#)

This walkthrough shows how to use Azure Data Lake to do data exploration and binary classification tasks on a sample of the NYC taxi trip and fare dataset. The sample shows you how to predict whether or not a tip is paid by a fare. It walks you through the steps of the [Team Data Science Process](#), end-to-end, from data acquisition to model training. Then it shows you how to deploy a web service that publishes the model.

Technologies

These technologies are used in this walkthrough.

- Azure Data Lake Analytics
- U-SQL and Visual Studio
- Python
- Azure Machine Learning
- Scripts

Azure Data Lake Analytics

The [Microsoft Azure Data Lake](#) has all the capabilities required to make it easy for data scientists to store data of any size, shape and speed, and to conduct data processing, advanced analytics, and machine learning modeling with high scalability in a cost-effective way. You pay on a per-job basis, only when data is actually being processed. Azure Data Lake Analytics includes U-SQL, a language that blends the declarative nature of SQL with the expressive power of C#. U-SQL then provides a scalable distributed query capability. It enables you to process unstructured data by applying schema on read. You can also insert custom logic and user-defined functions (UDFs), and it includes extensibility to enable fine-grained control over how to execute at scale. To learn more about the design philosophy behind U-SQL, see [this Visual Studio blog post](#).

Data Lake Analytics is also a key part of Cortana Analytics Suite. It works with Azure Synapse Analytics, Power BI, and Data Factory. This combination gives you a complete cloud big data and advanced analytics platform.

This walkthrough begins by describing how to install the prerequisites and resources that you need to complete the data science process tasks. Then it outlines the data processing steps using U-SQL and concludes by showing how to use Python and Hive with Azure Machine Learning studio (classic) to build and deploy the predictive models.

U-SQL and Visual Studio

This walkthrough recommends using Visual Studio to edit U-SQL scripts to process the dataset. The U-SQL scripts are described here and provided in a separate file. The process includes ingesting, exploring, and sampling the data. It also shows how to run a U-SQL scripted job from the Azure portal. Hive tables are created for the data in an associated HDInsight cluster to facilitate the building and deployment of a binary classification model in Azure Machine Learning studio.

Python

This walkthrough also contains a section that shows how to build and deploy a predictive model using Python with Azure Machine Learning studio. It provides a Jupyter Notebook with the Python scripts for the steps in this process. The notebook includes code for some additional feature engineering steps and models construction such as multiclass classification and regression modeling in addition to the binary classification model outlined here. The regression task is to predict the amount of the tip based on other tip features.

Azure Machine Learning

Azure Machine Learning studio (classic) is used to build and deploy the predictive models using two approaches: first with Python scripts and then with Hive tables on an HDInsight (Hadoop) cluster.

Scripts

Only the principal steps are outlined in this walkthrough. You can download the full U-SQL script and Jupyter Notebook from [GitHub](#).

Prerequisites

Before you begin these topics, you must have the following:

- An Azure subscription. If you don't already have one, see [Get Azure free trial](#).
- [Recommended] Visual Studio 2013 or later. If you don't already have one of these versions installed, you can download a free Community version from [Visual Studio Community](#).

NOTE

Instead of Visual Studio, you can also use the Azure portal to submit Azure Data Lake queries. Instructions are provided on how to do so both with Visual Studio and on the portal in the section titled [Process data with U-SQL](#).

Prepare data science environment for Azure Data Lake

To prepare the data science environment for this walkthrough, create the following resources:

- Azure Data Lake Storage (ADLS)
- Azure Data Lake Analytics (ADLA)
- Azure Blob storage account
- Azure Machine Learning studio (classic) account
- Azure Data Lake Tools for Visual Studio (Recommended)

This section provides instructions on how to create each of these resources. If you choose to use Hive tables with Azure Machine Learning, instead of Python, to build a model, you also need to provision an HDInsight (Hadoop)

cluster. This alternative procedure is described in the Option 2 section.

NOTE

The Azure Data Lake Store can be created either separately or when you create the Azure Data Lake Analytics as the default storage. Instructions are referenced for creating each of these resources separately, but the Data Lake storage account need not be created separately.

Create an Azure Data Lake Storage

Create an ADLS from the [Azure portal](#). For details, see [Create an HDInsight cluster with Data Lake Store using Azure portal](#). Be sure to set up the Cluster AAD Identity in the **DataSource** blade of the **Optional Configuration** blade described there.

The screenshot shows the Microsoft Azure portal interface. On the left, the 'New' button is highlighted with a red box. In the center, under the 'Data + Storage' category, the 'Data Lake Store' option is highlighted with a red box. The 'Data Lake Store' card describes it as a 'Hyper-scale repository for big data analytic workloads'.

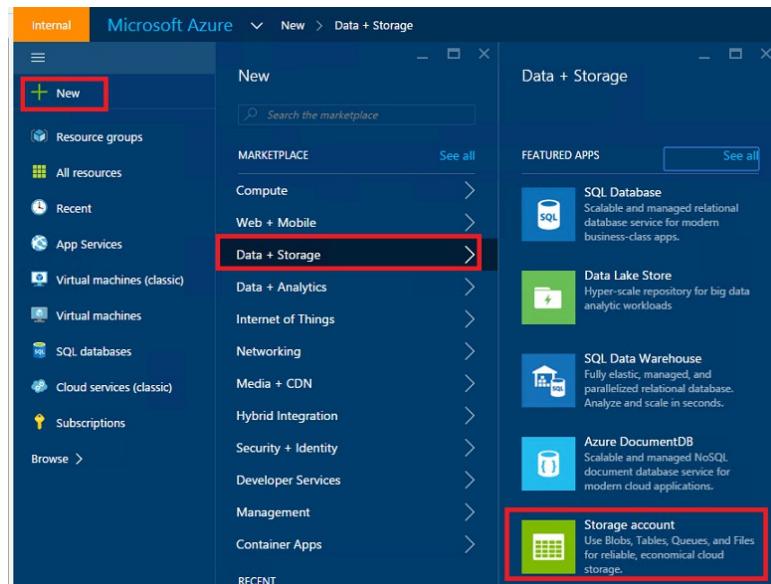
Create an Azure Data Lake Analytics account

Create an ADLA account from the [Azure portal](#). For details, see [Tutorial: get started with Azure Data Lake Analytics using Azure portal](#).

The screenshot shows the Microsoft Azure portal interface. On the left, the 'New' button is highlighted with a red box. In the center, under the 'Data + Analytics' category, the 'Data Lake Analytics' option is highlighted with a red box. The 'Data Lake Analytics' card describes it as 'Big data analytics made easy'.

Create an Azure Blob storage account

Create an Azure Blob storage account from the [Azure portal](#). For details, see the [Create a storage account](#) section in [About Azure Storage accounts](#).



Set up an Azure Machine Learning studio (classic) account

Sign up/into Azure Machine Learning studio (classic) from the [Azure Machine Learning studio](#) page. Click on the **Get started now** button and then choose a "Free Workspace" or "Standard Workspace". Now you are ready to create experiments in Azure Machine Learning studio.

Install Azure Data Lake Tools [Recommended]

Install Azure Data Lake Tools for your version of Visual Studio from [Azure Data Lake Tools for Visual Studio](#).

Azure Data Lake Tools for Visual Studio

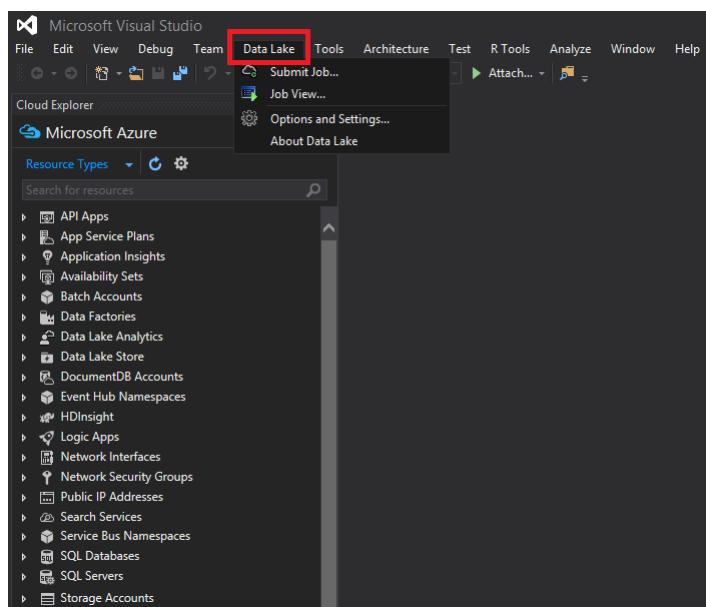
Language: English

Download

Plug-in for Azure Data Lake development using Visual Studio

- [Details](#)
- [System Requirements](#)
- [Install Instructions](#)

After the installation finishes, open up Visual Studio. You should see the Data Lake tab the menu at the top. Your Azure resources should appear in the left panel when you sign into your Azure account.



The NYC Taxi Trips dataset

The data set used here is a publicly available dataset -- the [NYC Taxi Trips dataset](#). The NYC Taxi Trip data consists of about 20 GB of compressed CSV files (~48 GB uncompressed), recording more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and dropoff locations and times, anonymized hack (driver's) license number, and the medallion (taxi's unique ID) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

The 'trip_data' CSV contains trip details, such as number of passengers, pickup and dropoff points, trip duration, and trip length. Here are a few sample records:

medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longitude
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01
15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06
00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.750666
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05
18:54:23,1,282,1.10,-74.004767,40.737777,-74.009834,40.726802
DFD2202EE08F7A8DC9A57B02AC881FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07
23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388
DFD2202EE08F7A8DC9A57B02AC881FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07
23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868

The 'trip_fare' CSV contains details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

medallion, hack_license, vendor_id, pickup_datetime, payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, total_amount
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0.5,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0,7
DFD2202EE08F7A8DC9A57B02AC881FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0.5,0.5,0,0,6
DFD2202EE08F7A8DC9A57B02AC881FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0,0.5,0,0,10.5

The unique key to join trip_data and trip_fare is composed of the following three fields: medallion, hack_license and pickup_datetime. The raw CSV files can be accessed from an Azure Storage blob. The U-SQL script for this join is in the [Join trip and fare tables](#) section.

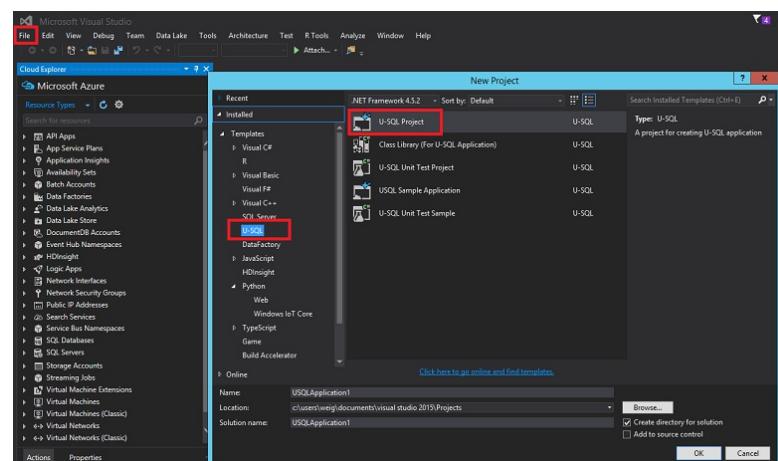
Process data with U-SQL

The data processing tasks illustrated in this section include ingesting, checking quality, exploring, and sampling the data. How to join trip and fare tables is also shown. The final section shows run a U-SQL scripted job from the Azure portal. Here are links to each subsection:

- [Data ingestion: read in data from public blob](#)
- [Data quality checks](#)
- [Data exploration](#)
- [Join trip and fare tables](#)
- [Data sampling](#)
- [Run U-SQL jobs](#)

The U-SQL scripts are described here and provided in a separate file. You can download the full U-SQL scripts from [GitHub](#).

To execute U-SQL, Open Visual Studio, click **File --> New --> Project**, choose **U-SQL Project**, name and save it to a folder.



NOTE

It's possible to use the Azure Portal to execute U-SQL instead of Visual Studio. You can navigate to the Azure Data Lake Analytics resource on the portal and submit queries directly as illustrated in the following figure:

```

1 // Replace weig with real values
2 DECLARE @par1 string = "weig";
3 DECLARE @outpath1 string = "/MLADS_Users/" + @par1 + "/task_55.csv";
4
5 //tipped vs. not tipped distribution
6 @tip_or_no =
7     SELECT *
8     FROM weig.dbo.fares
9
10 @task_5 =
11     SELECT tipped,
12     COUNT(*) AS tip_freq
13     FROM @tip_or_no
14     GROUP BY tipped;
15
16     OUTPUT @task_5
17     TO @outpath1
18     USING Outputters.Csv();

```

Data Ingestion: Read in data from public blob

The location of the data in the Azure blob is referenced as `wasb://container_name@blob_storage_account_name.blob.core.windows.net/blob_name` and can be extracted using `Extractors.Csv()`. Substitute your own container name and storage account name in following scripts for `container_name@blob_storage_account_name` in the wasb address. Since the file names are in same format, it's possible to use `trip_data_{*}.csv` to read in all 12 trip files.

```
//Read in Trip data
@trip0 =
    EXTRACT
    medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    passenger_count string,
    trip_time_in_secs string,
    trip_distance string,
    pickup_longitude string,
    pickup_latitude string,
    dropoff_longitude string,
    dropoff_latitude string
// This is reading 12 trip data from blob
FROM "wasb://container_name@blob_storage_account_name.blob.core.windows.net/nyctaxitrip/trip_data_{*}.csv"
USING Extractors.Csv();
```

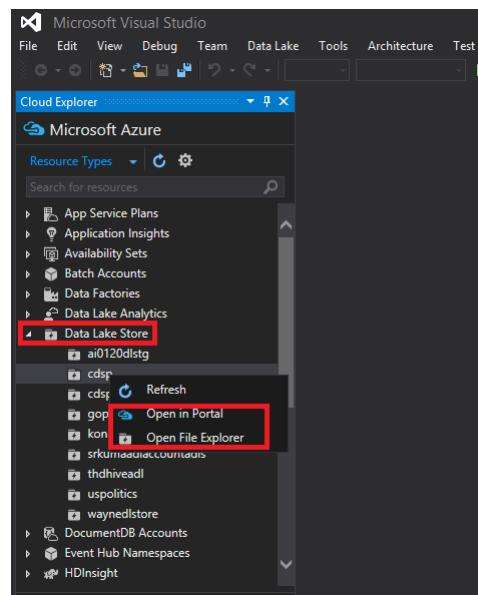
Since there are headers in the first row, you need to remove the headers and change column types into appropriate ones. You can either save the processed data to Azure Data Lake Storage using `swebhdfs://data_lake_storage_name.azuredatastorage.net/folder_name/file_name_` or to Azure Blob storage account using `wasb://container_name@blob_storage_account_name.blob.core.windows.net/blob_name`.

```
// change data types
@trip =
    SELECT
    medallion,
    hack_license,
    vendor_id,
    rate_code,
    store_and_fwd_flag,
    DateTime.Parse(pickup_datetime) AS pickup_datetime,
    DateTime.Parse(dropoff_datetime) AS dropoff_datetime,
    Int32.Parse(passenger_count) AS passenger_count,
    Double.Parse(trip_time_in_secs) AS trip_time_in_secs,
    Double.Parse(trip_distance) AS trip_distance,
    (pickup_longitude==string.Empty ? 0: float.Parse(pickup_longitude)) AS pickup_longitude,
    (pickup_latitude==string.Empty ? 0: float.Parse(pickup_latitude)) AS pickup_latitude,
    (dropoff_longitude==string.Empty ? 0: float.Parse(dropoff_longitude)) AS dropoff_longitude,
    (dropoff_latitude==string.Empty ? 0: float.Parse(dropoff_latitude)) AS dropoff_latitude
FROM @trip0
WHERE medallion != "medallion";

////output data to ADL
OUTPUT @trip
TO "swebhdfs://data_lake_storage_name.azuredatastorage.net/nyctaxi_folder/demo_trip.csv"
USING Outputters.Csv();

////Output data to blob
OUTPUT @trip
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_trip.csv"
USING Outputters.Csv();
```

Similarly you can read in the fare data sets. Right-click Azure Data Lake Storage, you can choose to look at your data in **Azure portal --> Data Explorer** or **File Explorer** within Visual Studio.



NAME	SIZE	LAST MODIFIED
trip0test.csv		4/12/2016, 2:57:42 PM
demo_ex_7_full_data.csv	37.6 GB	4/13/2016, 7:35:22 AM
demo_ex_9_stratified_1_1000_copy.csv	39.1 MB	4/13/2016, 3:39:07 AM
demo_ex_9_stratified_1_1000.csv	39.1 MB	4/13/2016, 7:35:25 AM
demo_fare.csv	22.5 GB	4/13/2016, 7:35:17 AM
demo_trip.csv	33.2 GB	4/13/2016, 7:35:15 AM
trip0_test.csv	2.87 GB	4/12/2016, 2:51:48 PM

Data quality checks

After trip and fare tables have been read in, data quality checks can be done in the following way. The resulting CSV files can be output to Azure Blob storage or Azure Data Lake Storage.

Find the number of medallions and unique number of medallions:

```
//check the number of medallions and unique number of medallions
@trip2 =
    SELECT
        medallion,
        vendor_id,
        pickup_datetime.Month AS pickup_month
    FROM @trip;

@ex_1 =
    SELECT
        pickup_month,
        COUNT(medallion) AS cnt_medallion,
        COUNT(DISTINCT(medallion)) AS unique_medallion
    FROM @trip
    GROUP BY pickup_month;
    OUTPUT @ex_1
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_1.csv"
    USING Outputters.Csv();
```

Find those medallions that had more than 100 trips:

```
//find those medallions that had more than 100 trips
@ex_2 =
    SELECT medallion,
        COUNT(medallion) AS cnt_medallion
    FROM @trip2
    WHERE pickup_datetime >= "2013-01-01T00:00:00.0000000" and pickup_datetime <= "2013-04-01T00:00:00.0000000"
    GROUP BY medallion
    HAVING COUNT(medallion) > 100;
    OUTPUT @ex_2
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_2.csv"
    USING Outputters.Csv();
```

Find those invalid records in terms of pickup_longitude:

```
//find those invalid records in terms of pickup_longitude
@ex_3 =
    SELECT COUNT(medallion) AS cnt_invalid_pickup_longitude
    FROM @trip
    WHERE
        pickup_longitude < -90 OR pickup_longitude > 90;
    OUTPUT @ex_3
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_3.csv"
    USING Outputters.Csv();
```

Find missing values for some variables:

```
//check missing values
@res =
    SELECT *,
        (medallion == null? 1 : 0) AS missing_medallion
    FROM @trip;

@trip_summary6 =
    SELECT
        vendor_id,
        SUM(missing_medallion) AS medallion_empty,
        COUNT(medallion) AS medallion_total,
        COUNT(DISTINCT(medallion)) AS medallion_total_unique
    FROM @res
    GROUP BY vendor_id;
    OUTPUT @trip_summary6
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_16.csv"
    USING Outputters.Csv();
```

Data exploration

Do some data exploration with the following scripts to get a better understanding of the data.

Find the distribution of tipped and non-tipped trips:

```

///tipped vs. not tipped distribution
@tip_or_not =
    SELECT *,
        (tip_amount > 0 ? 1: 0) AS tipped
    FROM @fare;

@ex_4 =
    SELECT tipped,
        COUNT(*) AS tip_freq
    FROM @tip_or_not
    GROUP BY tipped;
    OUTPUT @ex_4
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_4.csv"
USING Outputters.Csv();

```

Find the distribution of tip amount with cut-off values: 0, 5, 10, and 20 dollars.

```

//tip_class/range distribution
@tip_class =
    SELECT *,
        (tip_amount >20? 4: (tip_amount >10? 3:(tip_amount >5 ? 2:(tip_amount > 0 ? 1: 0)))) AS tip_class
    FROM @fare;
@ex_5 =
    SELECT tip_class,
        COUNT(*) AS tip_freq
    FROM @tip_class
    GROUP BY tip_class;
    OUTPUT @ex_5
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_5.csv"
USING Outputters.Csv();

```

Find basic statistics of trip distance:

```

// find basic statistics for trip_distance
@trip_summary4 =
    SELECT
        vendor_id,
        COUNT(*) AS cnt_row,
        MIN(trip_distance) AS min_trip_distance,
        MAX(trip_distance) AS max_trip_distance,
        AVG(trip_distance) AS avg_trip_distance
    FROM @trip
    GROUP BY vendor_id;
    OUTPUT @trip_summary4
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_14.csv"
USING Outputters.Csv();

```

Find the percentiles of trip distance:

```

// find percentiles of trip_distance
@trip_summary3 =
    SELECT DISTINCT vendor_id AS vendor,
        PERCENTILE_DISC(0.25) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id)
    AS median_trip_distance_disc,
        PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id)
    AS median_trip_distance_disc,
        PERCENTILE_DISC(0.75) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id)
    AS median_trip_distance_disc
    FROM @trip;
    // group by vendor_id;
    OUTPUT @trip_summary3
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_13.csv"
USING Outputters.Csv();

```

Join trip and fare tables

Trip and fare tables can be joined by medallion, hack_license, and pickup_time.

```

//join trip and fare table

@model_data_full =
    SELECT t.*,
        f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount,
        (f.tip_amount > 0 ? 1: 0) AS tipped,
        (f.tip_amount >20? 4: (f.tip_amount >10? 3:(f.tip_amount >5 ? 2:(f.tip_amount > 0 ? 1: 0)))) AS tip_class
    FROM @trip AS t JOIN @fare AS f
    ON (t.medallion == f.medallion AND t.hack_license == f.hack_license AND t.pickup_datetime ==
    f.pickup_datetime)
    WHERE (pickup_longitude != 0 AND dropoff_longitude != 0);

//// output to blob
OUTPUT @model_data_full
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_7_full_data.csv"
USING Outputters.Csv();

////output data to ADL
OUTPUT @model_data_full
TO "swebhdfs://data_lake_storage_name.azuredatastorage.net/nytaxi_folder/demo_ex_7_full_data.csv"
USING Outputters.Csv();

```

For each level of passenger count, calculate the number of records, average tip amount, variance of tip amount, percentage of tipped trips.

```

// contingency table
@trip_summary8 =
    SELECT passenger_count,
           COUNT(*) AS cnt,
           AVG(tip_amount) AS avg_tip_amount,
           VAR(tip_amount) AS var_tip_amount,
           SUM(tipped) AS cnt_tipped,
           (float)SUM(tipped)/COUNT(*) AS pct_tipped
    FROM @model_data_full
   GROUP BY passenger_count;
   OUTPUT @trip_summary8
   TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_17.csv"
  USING Outputters.Csv();

```

Data sampling

First, randomly select 0.1% of the data from the joined table:

```

//random select 1/1000 data for modeling purpose
@addrownumeres_randomsample =
SELECT *,
       ROW_NUMBER() OVER() AS rownum
FROM @model_data_full;

@model_data_random_sample_1_1000 =
SELECT *
FROM @addrownumeres_randomsample
WHERE rownum % 1000 == 0;

OUTPUT @model_data_random_sample_1_1000
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_7_random_1_1000.csv"
USING Outputters.Csv();

```

Then do stratified sampling by binary variable tip_class:

```

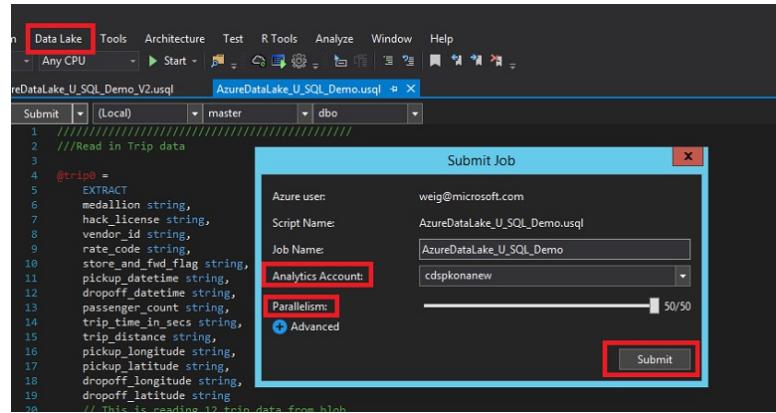
//stratified random select 1/1000 data for modeling purpose
@addrownumeres_stratifiedsample =
SELECT *,
       ROW_NUMBER() OVER(PARTITION BY tip_class) AS rownum
FROM @model_data_full;

@model_data_stratified_sample_1_1000 =
SELECT *
FROM @addrownumeres_stratifiedsample
WHERE rownum % 1000 == 0;
//// output to blob
OUTPUT @model_data_stratified_sample_1_1000
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();
////output data to ADL
OUTPUT @model_data_stratified_sample_1_1000
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nytaxi_folder/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();

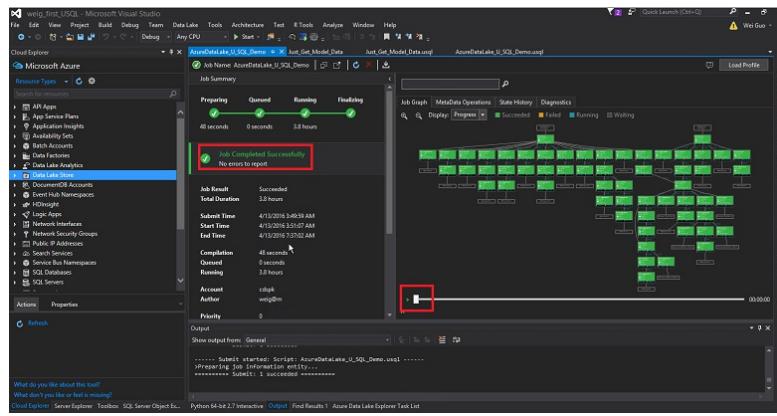
```

Run U-SQL jobs

After editing U-SQL scripts, you can submit them to the server using your Azure Data Lake Analytics account. Click **Data Lake**, **Submit Job**, select your **Analytics Account**, choose **Parallelism**, and click **Submit** button.



When the job is compiled successfully, the status of your job is displayed in Visual Studio for monitoring. After the job completes, you can even replay the job execution process and find out the bottleneck steps to improve your job efficiency. You can also go to Azure portal to check the status of your U-SQL jobs.



STATUS	JOB NAME	LANGUAGE
● Succeeded	AzureDataLake_JI_SQL_Demo	U-SQL
		U-SQL
		U-SQL
		U-SQL

Now you can check the output files in either Azure Blob storage or Azure portal. Use the stratified sample data for our modeling in the next step.

Name	Type	Last Modified	Length	Content Type
demo_ex_1.csv	Block	4/13/2016 2:35:23 PM +0:00	219 bytes	text/plain; charset=utf-8
demo_ex_10.csv	Block	4/13/2016 2:35:26 PM +0:00	40 bytes	text/plain; charset=utf-8
demo_ex_11.csv	Block	4/13/2016 2:35:18 PM +0:00	32 bytes	text/plain; charset=utf-8
demo_ex_12.csv	Block	4/13/2016 2:35:18 PM +0:00	118 bytes	text/plain; charset=utf-8
demo_ex_13.csv	Block	4/13/2016 2:35:20 PM +0:00	30 bytes	text/plain; charset=utf-8
demo_ex_14.csv	Block	4/13/2016 2:35:19 PM +0:00	89 bytes	text/plain; charset=utf-8
demo_ex_15.csv	Block	4/13/2016 2:35:20 PM +0:00	22 bytes	text/plain; charset=utf-8
demo_ex_16.csv	Block	4/13/2016 2:35:20 PM +0:00	46 bytes	text/plain; charset=utf-8
demo_ex_17.csv	Block	4/13/2016 2:35:24 PM +0:00	695 bytes	text/plain; charset=utf-8
demo_ex_2.csv	Block	4/13/2016 2:35:21 PM +0:00	550.96K	text/plain; charset=utf-8
demo_ex_3.csv	Block	4/13/2016 2:35:19 PM +0:00	6 bytes	text/plain; charset=utf-8
demo_ex_4.csv	Block	4/13/2016 2:35:19 PM +0:00	24 bytes	text/plain; charset=utf-8
demo_ex_5.csv	Block	4/13/2016 2:35:19 PM +0:00	55 bytes	text/plain; charset=utf-8
demo_ex_6.csv	Block	4/13/2016 2:35:19 PM +0:00	187.02K	text/plain; charset=utf-8
demo_ex_7_full_data.csv	Block	4/13/2016 2:35:22 PM +0:00	35.05M	text/plain; charset=utf-8
demo_ex_7_random_1_1000.csv	Block	4/13/2016 2:35:24 PM +0:00	37.41M	text/plain; charset=utf-8
demo_ex_8.csv	Block	4/13/2016 2:35:27 PM +0:00	40 bytes	text/plain; charset=utf-8
demo_ex_9_stratified_1_1000.csv	Block	4/13/2016 2:35:26 PM +0:00	37.32M	text/plain; charset=utf-8
demo_ex_9_stratified_1_1000_copy.csv	Block	4/13/2016 10:39:08 AM +0:00	37.32M	text/plain; charset=utf-8
demo_fare.csv	Block	4/13/2016 2:35:18 PM +0:00	20.97G	text/plain; charset=utf-8
demo_trip.csv	Block	4/13/2016 2:35:16 PM +0:00	30.88G	text/plain; charset=utf-8

NAME	SIZE	LAST MODIFIED
trip0test.csv		4/12/2016, 2:57:42 PM
demo_ex_7_full_data.csv	37.6 GB	4/13/2016, 7:35:22 AM
demo_ex_9_stratified_1_1000_copy.csv	39.1 MB	4/13/2016, 3:39:07 AM
demo_ex_9_stratified_1_1000.csv	39.1 MB	4/13/2016, 7:35:25 AM
demo_fare.csv	22.5 GB	4/13/2016, 7:35:17 AM
demo_trip.csv	33.2 GB	4/13/2016, 7:35:15 AM
trip0_test.csv	2.87 GB	4/12/2016, 2:51:48 PM

Build and deploy models in Azure Machine Learning

Two options are available for you to pull data into Azure Machine Learning to build and

- In the first option, you use the sampled data that has been written to an Azure Blob (in the **Data sampling** step above) and use Python to build and deploy models from Azure Machine Learning.
- In the second option, you query the data in Azure Data Lake directly using a Hive query. This option requires that you create a new HDInsight cluster or use an existing HDInsight cluster where the Hive tables point to the NY Taxi data in Azure Data Lake Storage. Both these options are discussed in the following sections.

Option 1: Use Python to build and deploy machine learning models

To build and deploy machine learning models using Python, create a Jupyter Notebook on your local machine or in Azure Machine Learning studio. The Jupyter Notebook provided on [GitHub](#) contains the full code to explore, visualize data, feature engineering, modeling, and deployment. In this article, just the modeling and deployment are covered.

Import Python libraries

In order to run the sample Jupyter Notebook or the Python script file, you need the following Python packages. If you're using the Azure Machine Learning Notebook service, these packages have been pre-installed.

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
import matplotlib.pyplot as plt
from time import time
import pyodbc
import os
from azure.storage.blob import BlobService
import tables
import time
import zipfile
import random
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from __future__ import division
from sklearn import linear_model
from azureml import services
```

Read in the data from blob

- Connection String

```
CONTAINERNAME = 'test1'
STORAGEACCOUNTNAME = 'XXXXXXXXXX'
STORAGEACCOUNTKEY = 'YYYYYYYYYYYYYYYYYYYYYYYYYYYY'
BLOBNAME = 'demo_ex_9_stratified_1_1000_copy.csv'
blob_service = BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
```

- Read in as text

```
t1 = time.time()
data = blob_service.get_blob_to_text(CONTAINERNAME,BLOBNAME).split("\n")
t2 = time.time()
print(("It takes %s seconds to read in "+BLOBNAME) % (t2 - t1))
```

It takes 1.61118912697 seconds to read in demo_ex_9_stratified_1_1000_copy.csv

- Add column names and separate columns

```
colnames =
['medallion','hack_license','vendor_id','rate_code','store_and_fwd_flag','pickup_datetime','dropoff_datetime',
'passenger_count','trip_time_in_secs','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude',
'dropoff_latitude','payment_type','fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount',
'tipped','tip_class','rownum']
df1 = pd.DataFrame([sub.split(",") for sub in data], columns = colnames)
```

- Change some columns to numeric

```
cols_2_float =
['trip_time_in_secs','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude',
'fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount',
'passenger_count','trip_distance',
'tipped','tip_class','rownum']
for col in cols_2_float:
    df1[col] = df1[col].astype(float)
```

Build machine learning models

Here you build a binary classification model to predict whether a trip is tipped or not. In the Jupyter Notebook you can find other two models: multiclass classification, and regression models.

- First you need to create dummy variables that can be used in scikit-learn models

```
df1_payment_type_dummy = pd.get_dummies(df1['payment_type'], prefix='payment_type_dummy')
df1_vendor_id_dummy = pd.get_dummies(df1['vendor_id'], prefix='vendor_id_dummy')
```

- Create data frame for the modeling

```
cols_to_keep = ['tipped', 'trip_distance', 'passenger_count']
data = df1[cols_to_keep].join([df1_payment_type_dummy,df1_vendor_id_dummy])

X = data.iloc[:,1:]
Y = data.tipped
```

- Training and testing 60-40 split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0)
```

- Logistic Regression in training set

```
model = LogisticRegression()
logit_fit = model.fit(X_train, Y_train)
print ('Coefficients: \n', logit_fit.coef_)
Y_train_pred = logit_fit.predict(X_train)
```

```
('Coefficients: \n', array([[-0.05133172, -0.008217 , 5.45355932, -6.63170116, -1.79058312,
 -2.79956737, 4.60387707, -0.33341385, -0.83100141]]))
```

- Score testing data set

```
Y_test_pred = logit_fit.predict(X_test)
```

- Calculate Evaluation metrics

```
fpr_train, tpr_train, thresholds_train = metrics.roc_curve(Y_train, Y_train_pred)
print fpr_train, tpr_train, thresholds_train

fpr_test, tpr_test, thresholds_test = metrics.roc_curve(Y_test, Y_test_pred)
print fpr_test, tpr_test, thresholds_test

#AUC
print metrics.auc(fpr_train,tpr_train)
print metrics.auc(fpr_test,tpr_test)

#Confusion Matrix
print metrics.confusion_matrix(Y_train,Y_train_pred)
print metrics.confusion_matrix(Y_test,Y_test_pred)
```

```
[ 0.         0.03372081  1.         ] [ 0.         0.99994397  1.         ] [ 2.         1.         0.         ]
[ 0.         0.03342893  1.         ] [ 0.         0.99983179  1.         ] [ 2.         1.         0.         ]
0.983111577209
0.983201427013
[[46966 1639]
 [ 3 53535]
 [[31343 1084]
 [ 6 35663]]]
```

Build Web Service API and consume it in Python

You want to operationalize the machine learning model after it has been built. The binary logistic model is used here as an example. Make sure the scikit-learn version in your local machine is 0.15.1 (Azure Machine Learning studio is already at least at this version).

- Find your workspace credentials from Azure Machine Learning studio (classic) settings. In Azure Machine Learning studio, click **Settings** --> **Name** --> **Authorization Tokens**.

```
workspaceid = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
auth_token = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

- Create Web Service

```
@services.publish(workspaceid, auth_token)
@services.types(trip_distance = float, passenger_count = float, payment_type_dummy_CRD = float,
payment_type_dummy_CSH=float, payment_type_dummy_DIS = float, payment_type_dummy_NOC = float,
payment_type_dummy_UNK = float, vendor_id_dummy_CMT = float, vendor_id_dummy_VTS = float)
@services.returns(int) #0, or 1
def predictNYCTAXI(trip_distance, passenger_count, payment_type_dummy_CRD,
payment_type_dummy_CSH,payment_type_dummy_DIS, payment_type_dummy_NOC, payment_type_dummy_UNK,
vendor_id_dummy_CMT, vendor_id_dummy_VTS ):
    inputArray = [trip_distance, passenger_count, payment_type_dummy_CRD, payment_type_dummy_CSH,
payment_type_dummy_DIS, payment_type_dummy_NOC, payment_type_dummy_UNK, vendor_id_dummy_CMT,
vendor_id_dummy_VTS]
    return logit_fit.predict(inputArray)
```

- Get web service credentials

```

url = predictNYCTAXI.service.url
api_key = predictNYCTAXI.service.api_key

print url
print api_key

@services.service(url, api_key)
@services.types(trip_distance = float, passenger_count = float, payment_type_dummy_CRD = float,
payment_type_dummy_CSH=float,payment_type_dummy_DIS = float, payment_type_dummy_NOC = float,
payment_type_dummy_UNL = float, vendor_id_dummy_CMT = float, vendor_id_dummy_VTS = float)
@services.returns(float)
def NYCTAXIPredictor(trip_distance, passenger_count, payment_type_dummy_CRD,
payment_type_dummy_CSH,payment_type_dummy_DIS, payment_type_dummy_NOC, payment_type_dummy_LINK,
vendor_id_dummy_CMT, vendor_id_dummy_VTS ):
    pass

```

- Call Web service API. Typically, wait 5-10 seconds after the previous step.

NYCTAXIPredictor(1,2,1,0,0,0,0,0,1)

1

Option 2: Create and deploy models directly in Azure Machine Learning

Azure Machine Learning studio (classic) can read data directly from Azure Data Lake Storage and then be used to create and deploy models. This approach uses a Hive table that points at the Azure Data Lake Storage. A separate Azure HDInsight cluster needs to be provisioned for the Hive table.

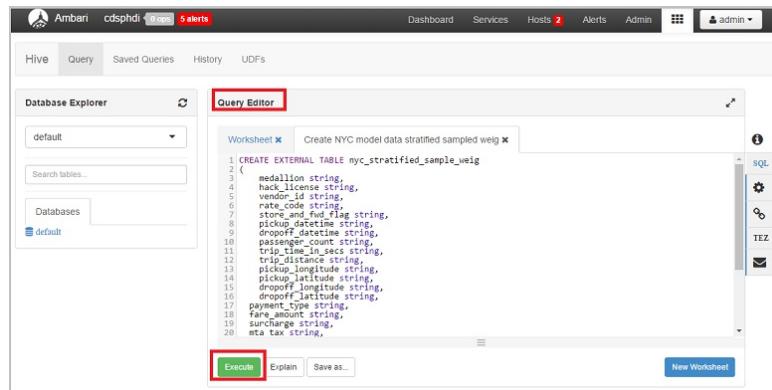
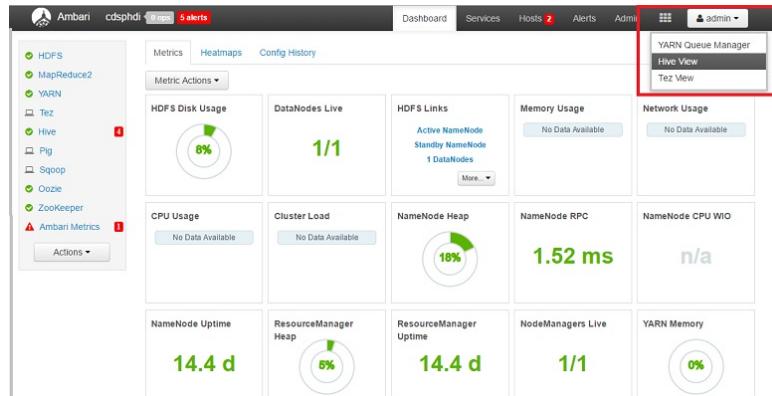
Create an HDInsight Linux Cluster

Create an HDInsight Cluster (Linux) from the [Azure portal](#). For details, see the [Create an HDInsight cluster with access to Azure Data Lake Storage](#) section in [Create an HDInsight cluster with Data Lake Store using Azure portal](#).

Create Hive table in HDInsight

Now you create Hive tables to be used in Azure Machine Learning studio (classic) in the HDInsight cluster using the data stored in Azure Data Lake Storage in the previous step. Go to the HDInsight cluster created. Click **Settings** --> **Properties** --> **Cluster AAD Identity** --> **ADLS Access**, make sure your Azure Data Lake Storage account is added in the list with read, write, and execute rights.

Then click **Dashboard** next to the **Settings** button and a window pops up. Click **Hive View** in the upper right corner of the page and you should see the **Query Editor**.



Paste in the following Hive scripts to create a table. The location of data source is in Azure Data Lake Storage reference in this way: `adl://data_lake_store_name.azuredatalakestore.net:443/folder_name/file_name`.

```

CREATE EXTERNAL TABLE nyc_stratified_sample
(
  medallion string,
  hack_license string,
  vendor_id string,
  rate_code string,
  store_and_fwd_flag string,
  pickup_datetime string,
  dropoff_datetime string,
  passenger_count string,
  trip_time_in_secs string,
  trip_distance string,
  pickup_longitude string,
  pickup_latitude string,
  dropoff_longitude string,
  dropoff_latitude string,
  payment_type string,
  fare_amount string,
  surcharge string,
  mta_tax string,
  tolls_amount string,
  total_amount string,
  tip_amount string,
  tipped string,
  tip_class string,
  rownum string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'
LOCATION
'adl://data_lake_store_name.azuredatalakestore.net:443/nytaxi_folder/demo_ex_9_stratified_1_1000_copy.csv';

```

When the query completes, you should see the results like this:

Query Process Results (Status: Succeeded)		
Logs	Results	Save results... ▾
Filter columns...		
nyc_stratified_sample_weig.medallion	nyc_stratified_sample_weig.hack_license	nyc_stratified_sample_weig
"005334C798EC6C8E637657962030F99"	"9EF690115D60940E7A8039A67542642E"	"VTS"
"00B99071EE4DC8266384113B91E6AC13"	"081B28EDA7F73E5E2C97573F0DBAC25D"	"CMT"
"011E4CBA1987A8553F8EA5681FFBF7F1"	"F53B26859F6C46C24E39EEAEE8096268"	"CMT"
"1109955CCAABCBC1A22BCED5F1DBFF5"	"EAA69F43239E5C6609CD8FF4D6725836"	"CMT"
"0A415B814A6479EE706972D2FD6DA08A"	"12A32F655B8C9CE3AC7872477A641974"	"VTS"
"02B29197FB7470B583ED12167D19E998"	"C1EEBC8298A619F86637D7E97F6BDD5C"	"CMT"
"03055B956C21B1F915DCDB118AA79F21"	"E0C7A67293DE535DB04F8AFD8BF28F73"	"VTS"
"037673EEAE0DCB912D06BED04E89D89D"	"3B247BD1230E95A0D9FED3E47FECB8D9"	"CMT"
"03BF54085C92C385889B957D804780D1"	"776D5633A2041CEBDE03092E401D61DB"	"CMT"
"0437660BC3704C2F185301D539434A64"	"AE9AB8C79A2A0EB6DDA76B7024FF6029"	"CMT"

Build and deploy models in Azure Machine Learning studio

You're now ready to build and deploy a model that predicts whether or not a tip is paid with Azure Machine Learning. The stratified sample data is ready to be used in this binary classification (tip or not) problem. The predictive models using multiclass classification (tip_class) and regression (tip_amount) can also be built and deployed with Azure Machine Learning studio, but here it's only shown how to handle the case using the binary classification model.

1. Get the data into Azure Machine Learning studio (classic) using the **Import Data** module, available in the **Data Input and Output** section. For more information, see the [Import Data module](#) reference page.

2. Select **Hive Query** as the **Data source** in the **Properties** panel.

3. Paste the following Hive script in the **Hive database query editor**

```
select * from nyc_stratified_sample;
```

4. Enter the URL of the HDInsight cluster (this URL can be found in the Azure portal), then enter the Hadoop credentials, the location of the output data, and the Azure Storage account name/key/container name.

Data source

```
1 select * from nyc_stratified_sample;
```

HCatalog server URI

Hadoop user account name

Hadoop user account password

Location of output data

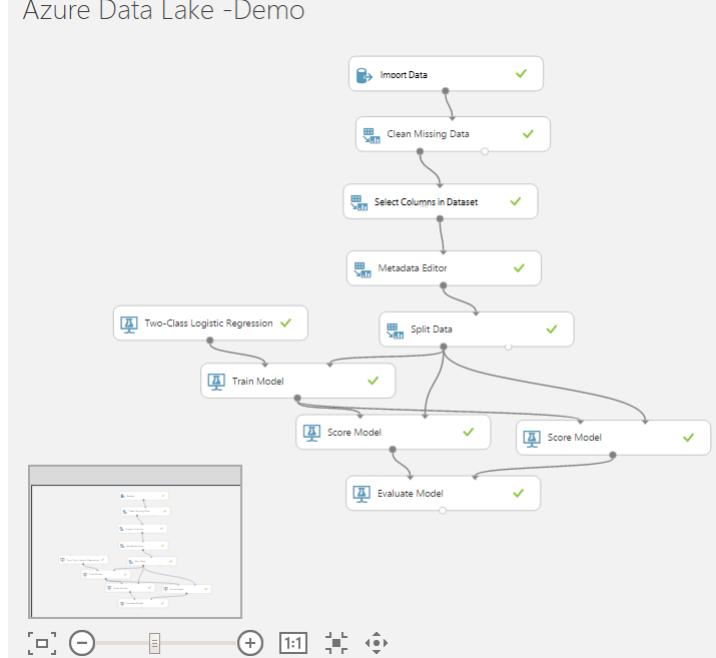
Azure storage account name

Azure storage key

Azure container name

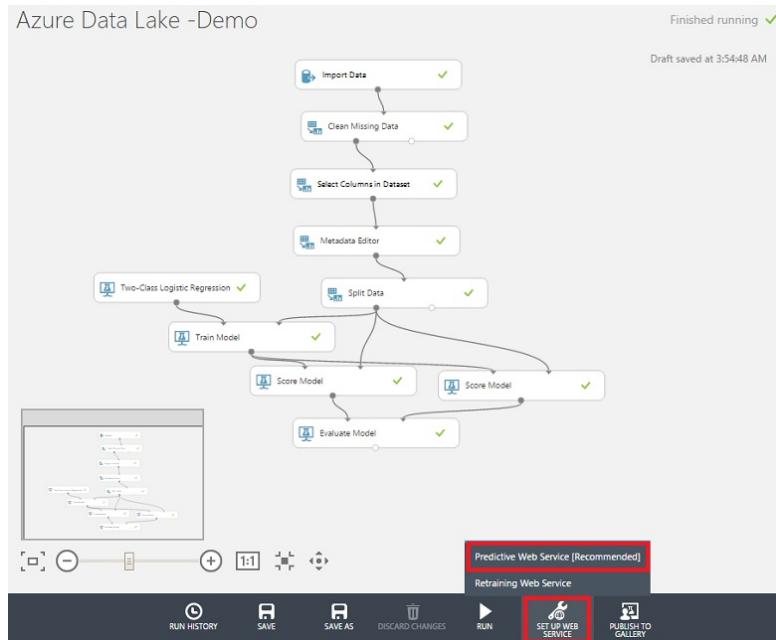
An example of a binary classification experiment reading data from Hive table is shown in the following figure:

Azure Data Lake -Demo

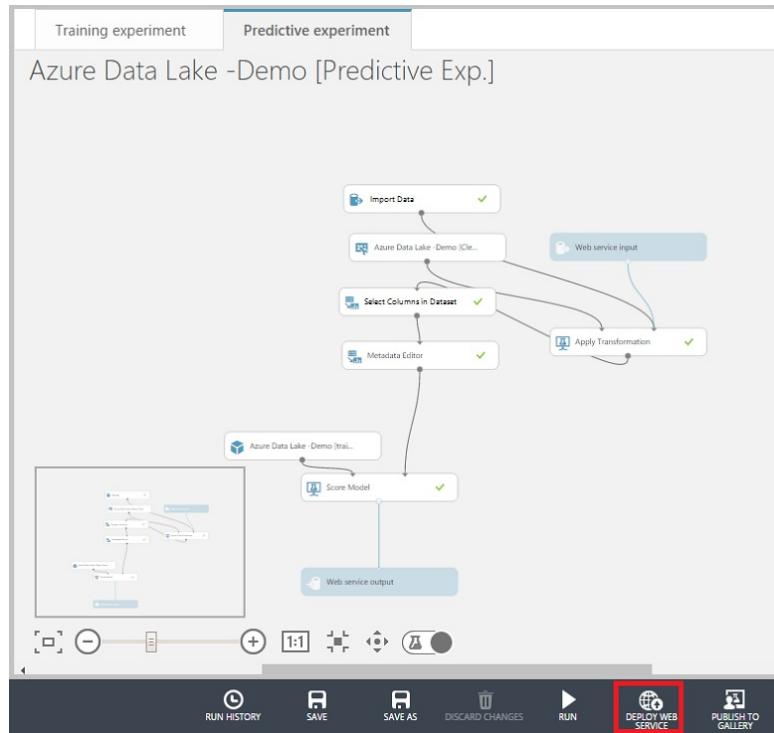


After the experiment is created, click **Set Up Web Service** --> **Predictive Web Service**

Azure Data Lake -Demo



Run the automatically created scoring experiment, when it finishes, click **Deploy Web Service**



The web service dashboard displays shortly:

Summary

By completing this walkthrough, you've created a data science environment for building scalable end-to-end solutions in Azure Data Lake. This environment was used to analyze a large public dataset, taking it through the canonical steps of the Data Science Process, from data acquisition through model training, and then to the deployment of the model as a web service. U-SQL was used to process, explore, and sample the data. Python and Hive were used with Azure Machine Learning studio (classic) to build and deploy predictive models.

Next steps

- [The Team Data Science Process in action: using Azure Synapse Analytics](#)
- [Overview of the Data Science Process using Spark on Azure HDInsight](#)

Related resources

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Process data in a SQL Server virtual machine on Azure

3/10/2022 • 6 minutes to read • [Edit Online](#)

This document covers how to explore data and generate features for data stored in a SQL Server VM on Azure. This goal may be completed by data wrangling using SQL or by using a programming language like Python.

NOTE

The sample SQL statements in this document assume that data is in SQL Server. If it isn't, refer to the cloud data science process map to learn how to move your data to SQL Server.

Using SQL

We describe the following data wrangling tasks in this section using SQL:

1. [Data Exploration](#)
2. [Feature Generation](#)

Data Exploration

Here are a few sample SQL scripts that can be used to explore data stores in SQL Server.

NOTE

For a practical example, you can use the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

1. Get the count of observations per day

```
SELECT CONVERT(date, <date_columnname>) as date, count(*) as c from <tablename> group by CONVERT(date, <date_columnname>)
```

2. Get the levels in a categorical column

```
select distinct <column_name> from <databasename>
```

3. Get the number of levels in combination of two categorical columns

```
select <column_a>, <column_b>, count(*) from <tablename> group by <column_a>, <column_b>
```

4. Get the distribution for numerical columns

```
select <column_name>, count(*) from <tablename> group by <column_name>
```

Feature Generation

In this section, we describe ways of generating features using SQL:

1. [Count based Feature Generation](#)
2. [Binning Feature Generation](#)
3. [Rolling out the features from a single column](#)

NOTE

Once you generate additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, that can be joined with the original table.

Count based Feature Generation

The following examples demonstrate two ways of generating count features. The first method uses conditional sum and the second method uses the 'where' clause. These results may then be joined with the original table (using primary key columns) to have count features alongside the original data.

```
select <column_name1>,<column_name2>,<column_name3>, COUNT(*) as Count_Features from <tablename> group by  
<column_name1>,<column_name2>,<column_name3>  
  
select <column_name1>,<column_name2> , sum(1) as Count_Features from <tablename>  
where <column_name3> = '<some_value>' group by <column_name1>,<column_name2>
```

Binning Feature Generation

The following example shows how to generate binned features by binning (using five bins) a numerical column that can be used as a feature instead:

```
SELECT <column_name>, NTILE(5) OVER (ORDER BY <column_name>) AS BinNumber from <tablename>
```

Rolling out the features from a single column

In this section, we demonstrate how to roll out a single column in a table to generate additional features. The example assumes that there is a latitude or longitude column in the table from which you are trying to generate features.

Here is a brief primer on latitude/longitude location data (resourced from stackoverflow [How to measure the accuracy of latitude and longitude?](#)). This guidance is useful to understand before including location as one or more features:

- The sign tells us whether we are north or south, east or west on the globe.
- A nonzero hundreds digit tells us that we're using longitude, not latitude!
- The tens digit gives a position to about 1,000 kilometers. It gives us useful information about what continent or ocean we are on.
- The units digit (one decimal degree) gives a position up to 111 kilometers (60 nautical miles, about 69 miles). It can tell you roughly what state, country, or region you're in.
- The first decimal place is worth up to 11.1 km: it can distinguish the position of one large city from a neighboring large city.
- The second decimal place is worth up to 1.1 km: it can separate one village from the next.
- The third decimal place is worth up to 110 m: it can identify a large agricultural field or institutional campus.
- The fourth decimal place is worth up to 11 m: it can identify a parcel of land. It is comparable to the typical accuracy of an uncorrected GPS unit with no interference.
- The fifth decimal place is worth up to 1.1 m: it distinguishes trees from each other. Accuracy to this level with commercial GPS units can only be achieved with differential correction.
- The sixth decimal place is worth up to 0.11 m: you can use this for laying out structures in detail, for designing landscapes, building roads. It should be more than good enough for tracking movements of glaciers and rivers. This can be achieved by taking painstaking measures with GPS, such as differentially corrected GPS.

The location information can be featurized as follows, separating out region, location, and city information. You

can also call a REST end point such as Bing Maps API available at [Find a Location by Point](#) to get the region/district information.

```
select
    <location_columnname>
    ,round(<location_columnname>,0) as l1
    ,l2=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 1 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,1) else '0' end
    ,l3=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 2 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,2) else '0' end
    ,l4=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 3 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,3) else '0' end
    ,l5=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 4 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,4) else '0' end
    ,l6=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 5 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,5) else '0' end
    ,l7=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 6 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,6) else '0' end
from <tablename>
```

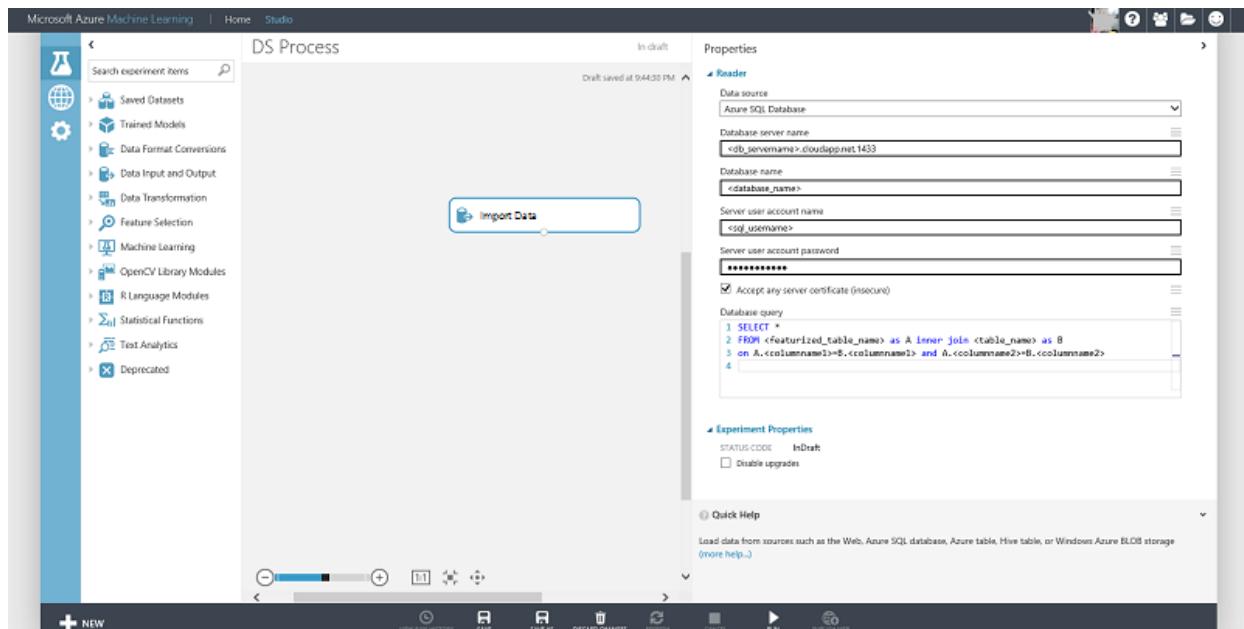
These location-based features can be further used to generate additional count features as described earlier.

TIP

You can programmatically insert the records using your language of choice. You may need to insert the data in chunks to improve write efficiency (for an example of how to do this using pyodbc, see [A HelloWorld sample to access SQLServer with python](#)). Another alternative is to insert data in the database using the [BCP utility](#).

Connecting to Azure Machine Learning

The newly generated feature can be added as a column to an existing table or stored in a new table and joined with the original table for machine learning. Features can be generated or accessed if already created, using the [Import Data](#) module in Azure Machine Learning as shown below:



Using a programming language like Python

Using Python to explore data and generate features when the data is in SQL Server is similar to processing data in Azure blob using Python as documented in [Process Azure Blob data in your data science environment](#). Load the data from the database into a pandas data frame for more processing. We document the process of connecting to the database and loading the data into the data frame in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replaceservername, dbname, username, and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The code below reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql('''select <columnname1>, <columnname2>... from <tablename>''', conn)
```

Now you can work with the Pandas data frame as covered in the article [Process Azure Blob data in your data science environment](#).

Azure Data Science in Action Example

For an end-to-end walkthrough example of the Azure Data Science Process using a public dataset, see [Azure Data Science Process in Action](#).

Cheat sheet for Azure Machine Learning designer

3/10/2022 • 2 minutes to read • [Edit Online](#)

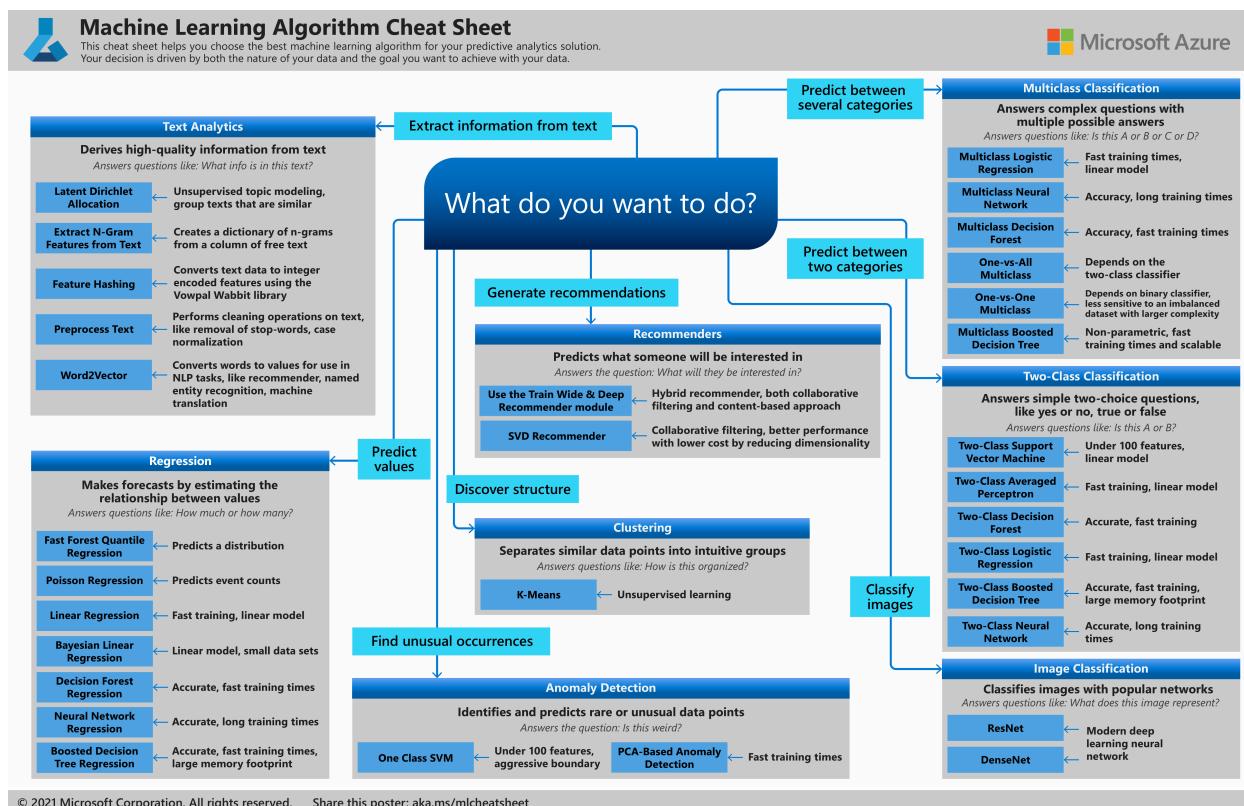
The **Azure Machine Learning Algorithm Cheat Sheet** helps you choose the right algorithm from the designer for a predictive analytics model.

Azure Machine Learning has a large library of algorithms from the classification, recommender systems, clustering, anomaly detection, regression, and text analytics families. Each is designed to address a different type of machine learning problem.

Download: Machine Learning Algorithm Cheat Sheet

Once you download the cheat sheet, you can print it in tabloid size (11 x 17 in.).

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)



Download and print the Machine Learning Algorithm Cheat Sheet in tabloid size to keep it handy and get help choosing an algorithm.

More help with Machine Learning

- For an overview of Microsoft Azure Machine Learning designer see [What is Azure Machine Learning designer?](#)
- For an overview of Microsoft Azure Machine Learning, see [What is Azure Machine Learning?](#).
- For an explanation of how to deploy a scoring web service, see [Deploy machine learning models to Azure](#).
- For a discussion of how to consume a scoring web service, see [Consume an Azure Machine Learning model deployed as a web service](#).

Data science using Spark on Azure HDInsight

3/10/2022 • 8 minutes to read • [Edit Online](#)

This suite of topics shows how to use HDInsight Spark to complete common data science tasks such as data ingestion, feature engineering, modeling, and model evaluation. The data used is a sample of the 2013 NYC taxi trip and fare dataset. The models built include logistic and linear regression, random forests, and gradient boosted trees. The topics also show how to store these models in Azure blob storage (WASB) and how to score and evaluate their predictive performance. More advanced topics cover how models can be trained using cross-validation and hyper-parameter sweeping. This overview topic also references the topics that describe how to set up the Spark cluster that you need to complete the steps in the walkthroughs provided.

Spark and MLlib

[Spark](#) is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for the iterative algorithms used in machine learning and graph computations. [MLlib](#) is Spark's scalable machine learning library that brings the algorithmic modeling capabilities to this distributed environment.

HDInsight Spark

[HDInsight Spark](#) is the Azure hosted offering of open-source Spark. It also includes support for [Jupyter PySpark notebooks](#) on the Spark cluster that can run Spark SQL interactive queries for transforming, filtering, and visualizing data stored in Azure Blobs (WASB). PySpark is the Python API for Spark. The code snippets that provide the solutions and show the relevant plots to visualize the data here run in Jupyter notebooks installed on the Spark clusters. The modeling steps in these topics contain code that shows how to train, evaluate, save, and consume each type of model.

Setup: Spark clusters and Jupyter notebooks

Setup steps and code are provided in this walkthrough for using an HDInsight Spark 1.6. But Jupyter notebooks are provided for both HDInsight Spark 1.6 and Spark 2.0 clusters. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked notebooks is generic and should work on any Spark cluster. If you are not using HDInsight Spark, the cluster setup and management steps may be slightly different from what is shown here. For convenience, here are the links to the Jupyter notebooks for Spark 1.6 (to be run in the pySpark kernel of the Jupyter Notebook server) and Spark 2.0 (to be run in the pySpark3 kernel of the Jupyter Notebook server):

Spark 1.6 notebooks

These notebooks are to be run in the pySpark kernel of Jupyter notebook server.

- [pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#): Provides information on how to perform data exploration, modeling, and scoring with several different algorithms.
- [pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): Includes topics in notebook #1, and model development using hyperparameter tuning and cross-validation.
- [pySpark-machine-learning-data-science-spark-model-consumption.ipynb](#): Shows how to operationalize a saved model using Python on HDInsight clusters.

Spark 2.0 notebooks

These notebooks are to be run in the pySpark3 kernel of Jupyter notebook server.

- [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): This file provides information on how to perform data exploration, modeling, and scoring in Spark 2.0 clusters using the NYC Taxi trip and fare data-set described [here](#). This notebook may be a good starting point for quickly exploring the code we have provided for Spark 2.0. For a more detailed notebook analyzes the NYC Taxi data, see the next notebook in this list. See the notes following this list that compares these notebooks.
- [Spark2.0-pySpark3_NYC_Taxi_Tip_Regression.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the NYC Taxi trip and fare data-set described [here](#).
- [Spark2.0-pySpark3_Airline_Departure_Delay_Classification.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the well-known Airline On-time departure dataset from 2011 and 2012. We integrated the airline dataset with the airport weather data (for example, windspeed, temperature, altitude etc.) prior to modeling, so these weather features can be included in the model.

NOTE

The airline dataset was added to the Spark 2.0 notebooks to better illustrate the use of classification algorithms. See the following links for information about airline on-time departure dataset and weather dataset:

- Airline on-time departure data: <https://www.transtats.bts.gov/ONTIME/>
- Airport weather data: <https://www.ncdc.noaa.gov/>

NOTE

The Spark 2.0 notebooks on the NYC taxi and airline flight delay data-sets can take 10 mins or more to run (depending on the size of your HDI cluster). The first notebook in the above list shows many aspects of the data exploration, visualization and ML model training in a notebook that takes less time to run with down-sampled NYC data set, in which the taxi and fare files have been pre-joined: [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#). This notebook takes a much shorter time to finish (2-3 mins) and may be a good starting point for quickly exploring the code we have provided for Spark 2.0.

For guidance on the operationalization of a Spark 2.0 model and model consumption for scoring, see the [Spark 1.6 document on consumption](#) for an example outlining the steps required. To use this example on Spark 2.0, replace the Python code file with [this file](#).

Prerequisites

The following procedures are related to Spark 1.6. For the Spark 2.0 version, use the notebooks described and linked to previously.

1. You must have an Azure subscription. If you do not already have one, see [Get Azure free trial](#).
 2. You need a Spark 1.6 cluster to complete this walkthrough. To create one, see the instructions provided in [Get started: create Apache Spark on Azure HDInsight](#). The cluster type and version is specified from the [Select Cluster Type](#) menu.

New HDInsight Cluster

*** Cluster Name**
Enter new cluster name .azurehdinsight.net

*** Subscription**
Azure-Irregulars_563702 >

Select Cluster Type **Spark (Preview)** (1) >

*** Credentials**
Configure required settings *

*** Data Source**
Configure required settings *

*** Node Pricing Tiers**
Configure required settings *

Pin to dashboard

Cluster Type configuration

Learn about HDInsight and cluster versions. [Learn more](#)

Cluster Type Spark (Preview)

Operating System Linux

Spark 1.5.2 (HDI 3.3)

Spark 1.6.0 (HDI 3.4)

Cluster Tier (more info)

STANDARD	PREMIUM (PREVIEW) *
[Administration Manage, monitor, connect [Scalability On-demand node scaling [99.9% Uptime SLA [Automatic patching	[Administration Manage, monitor, connect [Scalability On-demand node scaling [99.9% Uptime SLA [Automatic patching [Microsoft R Server for HDInsight
+ 0.00 USD/CORE/HOUR	+ 0.02 USD/CORE/HOUR

NOTE

For a topic that shows how to use Scala rather than Python to complete tasks for an end-to-end data science process, see the [Data Science using Scala with Spark on Azure](#).

WARNING

Billing for HDInsight clusters is prorated per minute, whether you use them or not. Be sure to delete your cluster after you finish using it. See [how to delete an HDInsight cluster](#).

The NYC 2013 Taxi data

The NYC Taxi Trip data is about 20 GB of compressed comma-separated values (CSV) files (~48 GB uncompressed), comprising more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and dropoff location and time, anonymized hack (driver's) license number and medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The 'trip_data' CSV files contain trip details, such as number of passengers, pick up and dropoff points, trip duration, and trip length. Here are a few sample records:

medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longit
89D227B655E5C82AEFC13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01 15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B678ED28BEA73D8,9FD8F69F0B040DBB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.750666
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0B040DBB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1.10,-74.004707,40.737777,-74.009834,40.726002
DFD2202EE08FTAB8DC9A57B02AC81FE2,51EE87E3205C985EF8431D0850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,,78,-73.974602,49.759945,-73.984734,40.759388
DFD2202EE08FTAB8DC9A57B02AC81FE2,51EE87E3205C985EF8431D0850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07 23:34:08,1,566,2,10,-73.97625,40.748528,-74.002586,40.747846

2. The 'trip_fare' CSV files contain details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

medallion, hack_license, vendor_id, pickup_datetime, payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, total_amount

89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01

0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-08 19:25:56,GH,5,2,5,2,2,2,7

0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804DB5549F40E9DA1BE472,CMT,2013-01-05
18:49:41,CSH,5.5,1,0.5,0,0,7

DDFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07
23:54:15,CSH,5,0.5,0.5,0,0,6

DDFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07
23:25:03,CSH,9.5,0.5,0.5,0,0,10.5

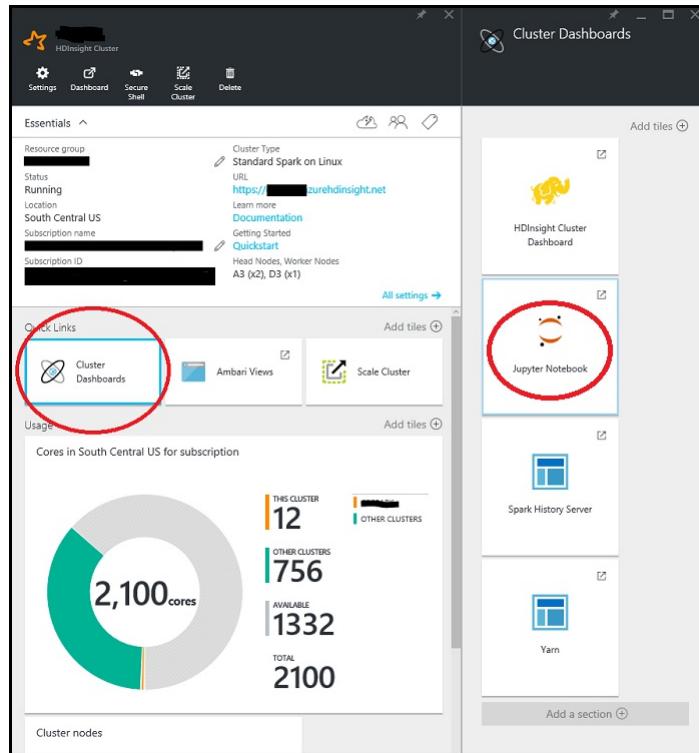
We have taken a 0.1% sample of these files and joined the trip_data and trip_fare CVS files into a single dataset to use as the input dataset for this walkthrough. The unique key to join trip_data and trip_fare is composed of the fields: medallion, hack_licence and pickup_datetime. Each record of the dataset contains the following

attributes representing a NYC Taxi trip:

FIELD	BRIEF DESCRIPTION
medallion	Anonymized taxi medallion (unique taxi id)
hack_license	Anonymized Hackney Carriage License number
vendor_id	Taxi vendor id
rate_code	NYC taxi rate of fare
store_and_fwd_flag	Store and forward flag
pickup_datetime	Pick up date & time
dropoff_datetime	Dropoff date & time
pickup_hour	Pick up hour
pickup_week	Pick up week of the year
weekday	Weekday (range 1-7)
passenger_count	Number of passengers in a taxi trip
trip_time_in_secs	Trip time in seconds
trip_distance	Trip distance traveled in miles
pickup_longitude	Pick up longitude
pickup_latitude	Pick up latitude
dropoff_longitude	Dropoff longitude
dropoff_latitude	Dropoff latitude
direct_distance	Direct distance between pickup and dropoff locations
payment_type	Payment type (cash, credit-card etc.)
fare_amount	Fare amount in
surcharge	Surcharge
mta_tax	MTA Metro Transportation tax
tip_amount	Tip amount
tolls_amount	Tolls amount
total_amount	Total amount
tipped	Tipped (0/1 for no or yes)
tip_class	Tip class (0: \$0, 1: \$0-5, 2: \$6-10, 3: \$11-20, 4: > \$20)

Execute code from a Jupyter notebook on the Spark cluster

You can launch the Jupyter Notebook from the Azure portal. Find your Spark cluster on your dashboard and click it to enter management page for your cluster. To open the notebook associated with the Spark cluster, click **Cluster Dashboards -> Jupyter Notebook**.



You can also browse to <https://CLUSTERNAME.azurehdinsight.net/jupyter> to access the Jupyter Notebooks. Replace the CLUSTERNAME part of this URL with the name of your own cluster. You need the password for your admin account to access the notebooks.

Select PySpark to see a directory that contains a few examples of pre-packaged notebooks that use the PySpark API. The notebooks that contain the code samples for this suite of Spark topic are available at [GitHub](#)

You can upload the notebooks directly from [GitHub](#) to the Jupyter notebook server on your Spark cluster. On the home page of your Jupyter, click the **Upload** button on the right part of the screen. It opens a file explorer. Here you can paste the GitHub (raw content) URL of the Notebook and click Open.

You see the file name on your Jupyter file list with an **Upload** button again. Click this **Upload** button. Now you have imported the notebook. Repeat these steps to upload the other notebooks from this walkthrough.

TIP

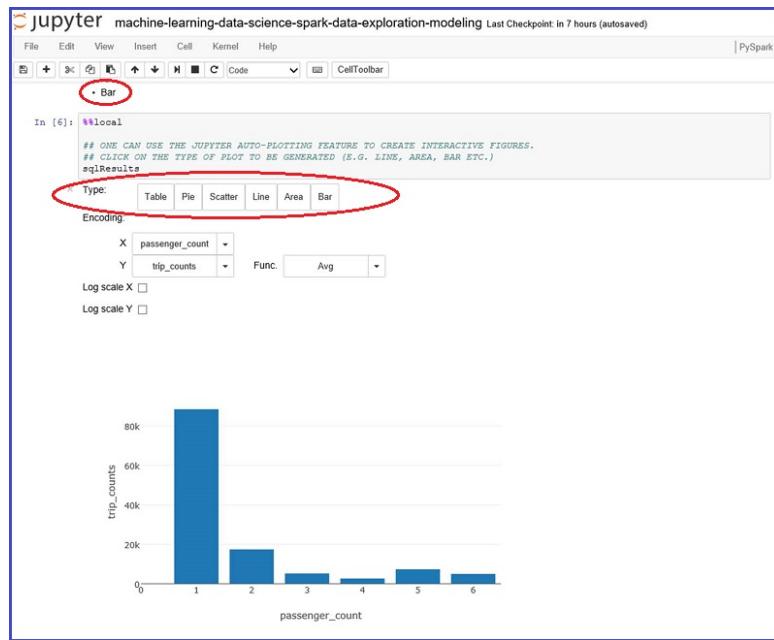
You can right-click the links on your browser and select **Copy Link** to get the GitHub raw content URL. You can paste this URL into the Jupyter Upload file explorer dialog box.

Now you can:

- See the code by clicking the notebook.
- Execute each cell by pressing **SHIFT-ENTER**.
- Run the entire notebook by clicking on **Cell -> Run**.
- Use the automatic visualization of queries.

TIP

The PySpark kernel automatically visualizes the output of SQL (HiveQL) queries. You are given the option to select among several different types of visualizations (Table, Pie, Line, Area, or Bar) by using the **Type** menu buttons in the notebook:



Next steps

- What is the Team Data Science Process?
- Compare the machine learning products and technologies from Microsoft
- Machine learning at scale

Data Science using Scala and Spark on Azure

3/10/2022 • 33 minutes to read • [Edit Online](#)

This article shows you how to use Scala for supervised machine learning tasks with the Spark scalable MLlib and Spark ML packages on an Azure HDInsight Spark cluster. It walks you through the tasks that constitute the [Data Science process](#): data ingestion and exploration, visualization, feature engineering, modeling, and model consumption. The models in the article include logistic and linear regression, random forests, and gradient-boosted trees (GBTs), in addition to two common supervised machine learning tasks:

- Regression problem: Prediction of the tip amount (\$) for a taxi trip
- Binary classification: Prediction of tip or no tip (1/0) for a taxi trip

The modeling process requires training and evaluation on a test data set and relevant accuracy metrics. In this article, you can learn how to store these models in Azure Blob storage and how to score and evaluate their predictive performance. This article also covers the more advanced topics of how to optimize models by using cross-validation and hyper-parameter sweeping. The data used is a sample of the 2013 NYC taxi trip and fare data set available on GitHub.

[Scala](#), a language based on the Java virtual machine, integrates object-oriented and functional language concepts. It's a scalable language that is well suited to distributed processing in the cloud, and runs on Azure Spark clusters.

[Spark](#) is an open-source parallel-processing framework that supports in-memory processing to boost the performance of big data analytics applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for iterative algorithms in machine learning and graph computations. The [spark.ml](#) package provides a uniform set of high-level APIs built on top of data frames that can help you create and tune practical machine learning pipelines. [MLlib](#) is Spark's scalable machine learning library, which brings modeling capabilities to this distributed environment.

[HDInsight Spark](#) is the Azure-hosted offering of open-source Spark. It also includes support for Jupyter Scala notebooks on the Spark cluster, and can run Spark SQL interactive queries to transform, filter, and visualize data stored in Azure Blob storage. The Scala code snippets in this article that provide the solutions and show the relevant plots to visualize the data run in Jupyter notebooks installed on the Spark clusters. The modeling steps in these topics have code that shows you how to train, evaluate, save, and consume each type of model.

The setup steps and code in this article are for Azure HDInsight 3.4 Spark 1.6. However, the code in this article and in the [Scala Jupyter Notebook](#) are generic and should work on any Spark cluster. The cluster setup and management steps might be slightly different from what is shown in this article if you are not using HDInsight Spark.

NOTE

For a topic that shows you how to use Python rather than Scala to complete tasks for an end-to-end Data Science process, see [Data Science using Spark on Azure HDInsight](#).

Prerequisites

- You must have an Azure subscription. If you do not already have one, [get an Azure free trial](#).
- You need an Azure HDInsight 3.4 Spark 1.6 cluster to complete the following procedures. To create a cluster,

see the instructions in [Get started: Create Apache Spark on Azure HDInsight](#). Set the cluster type and version on the Select Cluster Type menu.

The screenshot shows the 'New HDInsight Cluster' configuration page. On the left, there are several configuration steps: 'Cluster Name' (with placeholder 'Enter new cluster name'), 'Subscription' (selected as 'Azure-Irregulars_563702'), 'Select Cluster Type' (with a warning icon), 'Credentials', 'Data Source', and 'Node Pricing Tiers'. On the right, the 'Cluster Type configuration' section is expanded. It includes a link to 'Learn about HDInsight and cluster versions'. Under 'Cluster Type', 'Spark (Preview)' is selected. Under 'Operating System', 'Linux' is selected. Under 'Cluster Tier', 'Spark 1.6.0 (HDI 3.4)' is selected. Other tiers shown are 'STANDARD' and 'PREMIUM (PREVIEW)'. Both tiers include 'Administration', 'Scalability', 'Uptime SLA', and 'Automatic patching'. The 'PREMIUM (PREVIEW)' tier also includes 'Microsoft R Server for HDInsight'. Pricing is listed as '+ 0.00' for STANDARD and '+ 0.02' for PREMIUM (PREVIEW) USD/CORE/HOUR. A red circle highlights the 'Spark (Preview)' dropdown, another red circle highlights the 'Linux' button, and a third red circle highlights the 'Spark 1.6.0 (HDI 3.4)' button.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you use them or not. Be sure to delete your cluster after you finish using it. See [how to delete an HDInsight cluster](#).

For a description of the NYC taxi trip data and instructions on how to execute code from a Jupyter notebook on the Spark cluster, see the relevant sections in [Overview of Data Science using Spark on Azure HDInsight](#).

Execute Scala code from a Jupyter notebook on the Spark cluster

You can launch a Jupyter notebook from the Azure portal. Find the Spark cluster on your dashboard, and then click it to enter the management page for your cluster. Next, click **Cluster Dashboards**, and then click **Jupyter Notebook** to open the notebook associated with the Spark cluster.

The screenshot shows the Azure HDInsight Cluster dashboard. At the top, there are navigation icons for Settings, Dashboard, Secure Shell, Scale Cluster, and Delete. Below this is the 'Essentials' section, which includes a 'Resource group' dropdown, 'Status' (Running), 'Location' (South Central US), 'Subscription name' (redacted), 'Subscription ID' (redacted), and 'Cluster Type' (Standard Spark on Linux). It also displays the URL <https://<clustername>.azurehdinsight.net>, 'Learn more' links for Documentation and Quickstart, and information about Head Nodes, Worker Nodes (A3 (x2), D3 (x1)). A red circle highlights the 'Cluster Dashboards' tile in the 'Quick Links' section. To the right is a 'Cluster Dashboards' panel containing tiles for the HDInsight Cluster Dashboard, Jupyter Notebook (also highlighted with a red circle), Spark History Server, and Yarn.

You also can access Jupyter notebooks at <https://<clustername>.azurehdinsight.net/jupyter>. Replace *clustername* with the name of your cluster. You need the password for your administrator account to access the Jupyter notebooks.

The screenshot shows the Jupyter Notebook interface. At the top, there are tabs for Files, Running, and Clusters. Below this is a file explorer sidebar with a 'PySpark' folder and a 'Scala' folder, which is circled in red. At the bottom right of the interface is an 'Upload' button, also circled in red.

Select **Scala** to see a directory that has a few examples of prepackaged notebooks that use the PySpark API. The Exploration Modeling and Scoring using Scala.ipynb notebook that contains the code samples for this suite of Spark topics is available on [GitHub](#).

You can upload the notebook directly from GitHub to the Jupyter Notebook server on your Spark cluster. On your Jupyter home page, click the **Upload** button. In the file explorer, paste the GitHub (raw content) URL of the Scala notebook, and then click **Open**. The Scala notebook is available at the following URL:

[Exploration-Modeling-and-Scoring-using-Scala.ipynb](#)

Setup: Preset Spark and Hive contexts, Spark magics, and Spark

libraries

Preset Spark and Hive contexts

```
# SET THE START TIME
import java.util.Calendar
val beginningTime = Calendar.getInstance().getTime()
```

The Spark kernels that are provided with Jupyter notebooks have preset contexts. You don't need to explicitly set the Spark or Hive contexts before you start working with the application you are developing. The preset contexts are:

- `sc` for `SparkContext`
- `sqlContext` for `HiveContext`

Spark magics

The Spark kernel provides some predefined "magics," which are special commands that you can call with `%%`. Two of these commands are used in the following code samples.

- `%%local` specifies that the code in subsequent lines will be executed locally. The code must be valid Scala code.
- `%%sql -o <variable name>` executes a Hive query against `sqlContext`. If the `-o` parameter is passed, the result of the query is persisted in the `%%local` Scala context as a Spark data frame.

For more information about the kernels for Jupyter notebooks and their predefined "magics" that you call with `%%` (for example, `%%local`), see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Import libraries

Import the Spark, MLlib, and other libraries you'll need by using the following code.

```

# IMPORT SPARK AND JAVA LIBRARIES
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.functions._
import java.text.SimpleDateFormat
import java.util.Calendar
import sqlContext.implicits._
import org.apache.spark.sql.Row

# IMPORT SPARK SQL FUNCTIONS
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType, FloatType, DoubleType}
import org.apache.spark.sql.functions.rand

# IMPORT SPARK ML FUNCTIONS
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler, OneHotEncoder, VectorIndexer, Binarizer}
import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit, CrossValidator}
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel, RandomForestRegressor,
RandomForestRegressionModel, GBTRegressor, GBTRegressionModel}
import org.apache.spark.ml.classification.{LogisticRegression, LogisticRegressionModel,
RandomForestClassifier, Random ForestClassificationModel, GBTClassifier, GBTClassificationModel}
import org.apache.spark.ml.evaluation.{BinaryClassificationEvaluator, RegressionEvaluator,
MulticlassClassificationEvaluator}

# IMPORT SPARK MLLIB FUNCTIONS
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}
import org.apache.spark.mllib.regression.{LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel}
import org.apache.spark.mllib.tree.{GradientBoostedTrees, RandomForest}
import org.apache.spark.mllib.tree.configuration.BoostingStrategy
import org.apache.spark.mllib.tree.model.{GradientBoostedTreesModel, RandomForestModel, Predict}
import org.apache.spark.mllib.evaluation.{BinaryClassificationMetrics, MulticlassMetrics, RegressionMetrics}

# SPECIFY SQLCONTEXT
val sqlContext = new SQLContext(sc)

```

Data ingestion

The first step in the Data Science process is to ingest the data that you want to analyze. You bring the data from external sources or systems where it resides into your data exploration and modeling environment. In this article, the data you ingest is a joined 0.1% sample of the taxi trip and fare file (stored as a .tsv file). The data exploration and modeling environment is Spark. This section contains the code to complete the following series of tasks:

1. Set directory paths for data and model storage.
2. Read in the input data set (stored as a .tsv file).
3. Define a schema for the data and clean the data.
4. Create a cleaned data frame and cache it in memory.
5. Register the data as a temporary table in SQLContext.
6. Query the table and import the results into a data frame.

Set directory paths for storage locations in Azure Blob storage

Spark can read and write to Azure Blob storage. You can use Spark to process any of your existing data, and then store the results again in Blob storage.

To save models or files in Blob storage, you need to properly specify the path. Reference the default container attached to the Spark cluster by using a path that begins with `wasb:///`. Reference other locations by using `wasb://`.

The following code sample specifies the location of the input data to be read and the path to Blob storage that is

attached to the Spark cluster where the model will be saved.

```
# SET PATHS TO DATA AND MODEL FILE LOCATIONS
# INGEST DATA AND SPECIFY HEADERS FOR COLUMNS
val taxi_train_file =
sc.textFile("wasb://mllibwalkthroughs@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare
.Point1Pct.Train.tsv")
val header = taxi_train_file.first;

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THAT THE FINAL BACKSLASH IN THE PATH IS REQUIRED.
val modelDir = "wasb://user/remoteuser/NYCTaxi/Models/";
```

Import data, create an RDD, and define a data frame according to the schema

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE SCHEMA BASED ON THE HEADER OF THE FILE
val sqlContext = new SQLContext(sc)
val taxi_schema = StructType(
  Array(
    StructField("medallion", StringType, true),
    StructField("hack_license", StringType, true),
    StructField("vendor_id", StringType, true),
    StructField("rate_code", DoubleType, true),
    StructField("store_and_fwd_flag", StringType, true),
    StructField("pickup_datetime", StringType, true),
    StructField("dropoff_datetime", StringType, true),
    StructField("pickup_hour", DoubleType, true),
    StructField("pickup_week", DoubleType, true),
    StructField("weekday", DoubleType, true),
    StructField("passenger_count", DoubleType, true),
    StructField("trip_time_in_secs", DoubleType, true),
    StructField("trip_distance", DoubleType, true),
    StructField("pickup_longitude", DoubleType, true),
    StructField("pickup_latitude", DoubleType, true),
    StructField("dropoff_longitude", DoubleType, true),
    StructField("dropoff_latitude", DoubleType, true),
    StructField("direct_distance", StringType, true),
    StructField("payment_type", StringType, true),
    StructField("fare_amount", DoubleType, true),
    StructField("surcharge", DoubleType, true),
    StructField("mta_tax", DoubleType, true),
    StructField("tip_amount", DoubleType, true),
    StructField("tolls_amount", DoubleType, true),
    StructField("total_amount", DoubleType, true),
    StructField("tipped", DoubleType, true),
    StructField("tip_class", DoubleType, true)
  )
)

# CAST VARIABLES ACCORDING TO THE SCHEMA
val taxi_temp = (taxi_train_file.map(_.split("\t"))
  .filter((r) => r(0) != "medallion")
  .map(p => Row(p(0), p(1), p(2),
    p(3).toDouble, p(4), p(5), p(6), p(7).toDouble, p(8).toDouble, p(9).toDouble,
    p(10).toDouble,
    p(11).toDouble, p(12).toDouble, p(13).toDouble, p(14).toDouble, p(15).toDouble,
    p(16).toDouble,
    p(17), p(18), p(19).toDouble, p(20).toDouble, p(21).toDouble, p(22).toDouble,
    p(23).toDouble, p(24).toDouble, p(25).toDouble, p(26).toDouble)))

# CREATE AN INITIAL DATA FRAME AND DROP COLUMNS, AND THEN CREATE A CLEANED DATA FRAME BY FILTERING FOR
UNWANTED VALUES OR OUTLIERS
val taxi_train_df = sqlContext.createDataFrame(taxi_temp, taxi_schema)
```

```

val taxi_df_train_cleaned = (taxi_train_df.drop(taxi_train_df.col("medallion"))
    .drop(taxi_train_df.col("hack_license")).drop(taxi_train_df.col("store_and_fwd_flag"))
    .drop(taxi_train_df.col("pickup_datetime")).drop(taxi_train_df.col("dropoff_datetime"))
    .drop(taxi_train_df.col("pickup_longitude")).drop(taxi_train_df.col("pickup_latitude"))
    .drop(taxi_train_df.col("dropoff_longitude")).drop(taxi_train_df.col("dropoff_latitude"))
    .drop(taxi_train_df.col("surcharge")).drop(taxi_train_df.col("mta_tax"))
    .drop(taxi_train_df.col("direct_distance")).drop(taxi_train_df.col("tolls_amount"))
    .drop(taxi_train_df.col("total_amount")).drop(taxi_train_df.col("tip_class"))
    .filter("passenger_count > 0 and passenger_count < 8 AND payment_type in ('CSH', 'CRD') AND
tip_amount >= 0 AND tip_amount < 30 AND fare_amount >= 1 AND fare_amount < 150 AND trip_distance > 0 AND
trip_distance < 100 AND trip_time_in_secs > 30 AND trip_time_in_secs < 7200"));

# CACHE AND MATERIALIZE THE CLEANED DATA FRAME IN MEMORY
taxi_df_train_cleaned.cache()
taxi_df_train_cleaned.count()

# REGISTER THE DATA FRAME AS A TEMPORARY TABLE IN SQLCONTEXT
taxi_df_train_cleaned.registerTempTable("taxi_train")

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 8 seconds.

Query the table and import results in a data frame

Next, query the table for fare, passenger, and tip data; filter out corrupt and outlying data; and print several rows.

```

# QUERY THE DATA
val sqlStatement = """
    SELECT fare_amount, passenger_count, tip_amount, tipped
    FROM taxi_train
    WHERE passenger_count > 0 AND passenger_count < 7
        AND fare_amount > 0 AND fare_amount < 200
        AND payment_type in ('CSH', 'CRD')
        AND tip_amount > 0 AND tip_amount < 25
"""
val sqlResultsDF = sqlContext.sql(sqlStatement)

# SHOW ONLY THE TOP THREE ROWS
sqlResultsDF.show(3)

```

Output:

FARE_AMOUNT	PASSENGER_COUNT	TIP_AMOUNT	TIPPED
13.5	1.0	2.9	1.0
16.0	2.0	3.4	1.0
10.5	2.0	1.0	1.0

Data exploration and visualization

After you bring the data into Spark, the next step in the Data Science process is to gain a deeper understanding of the data through exploration and visualization. In this section, you examine the taxi data by using SQL queries. Then, import the results into a data frame to plot the target variables and prospective features for visual

inspection by using the automatic visualization Jupyter feature.

Use local and SQL magic to plot data

By default, the output of any code snippet that you run from a Jupyter notebook is available within the context of the session that is persisted on the worker nodes. If you want to save a trip to the worker nodes for every computation, and if all the data that you need for your computation is available locally on the Jupyter server node (which is the head node), you can use the `%%local` magic to run the code snippet on the Jupyter server.

- **SQL magic (`%sql`)**. The HDInsight Spark kernel supports easy inline HiveQL queries against `SQLContext`. The `(-o VARIABLE_NAME)` argument persists the output of the SQL query as a Pandas data frame on the Jupyter server. This setting means the output will be available in the local mode.
- **%%local magic**. The `%%local` magic runs the code locally on the Jupyter server, which is the head node of the HDInsight cluster. Typically, you use `%%local` magic in conjunction with the `%sql` magic with the `-o` parameter. The `-o` parameter would persist the output of the SQL query locally, and then `%%local` magic would trigger the next set of code snippet to run locally against the output of the SQL queries that is persisted locally.

Query the data by using SQL

This query retrieves the taxi trips by fare amount, passenger count, and tip amount.

```
# RUN THE SQL QUERY
%%sql -q -o sqlResults
SELECT fare_amount, passenger_count, tip_amount, tipped FROM taxi_train WHERE passenger_count > 0 AND
passenger_count < 7 AND fare_amount > 0 AND fare_amount < 200 AND payment_type in ('CSH', 'CRD') AND
tip_amount > 0 AND tip_amount < 25
```

In the following code, the `%%local` magic creates a local data frame, `sqlResults`. You can use `sqlResults` to plot by using `matplotlib`.

TIP

Local magic is used multiple times in this article. If your data set is large, please sample to create a data frame that can fit in local memory.

Plot the data

You can plot by using Python code after the data frame is in local context as a Pandas data frame.

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES.
# CLICK THE TYPE OF PLOT TO GENERATE (LINE, AREA, BAR, ETC.)
sqlResults
```

The Spark kernel automatically visualizes the output of SQL (HiveQL) queries after you run the code. You can choose between several types of visualizations:

- Table
- Pie
- Line
- Area
- Bar

Here's the code to plot the data:

```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
import matplotlib.pyplot as plt
%matplotlib inline

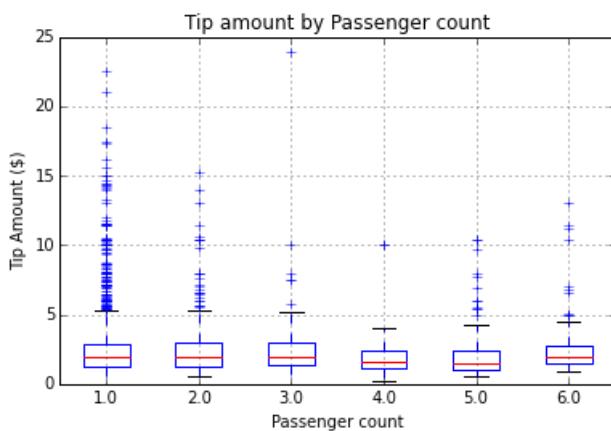
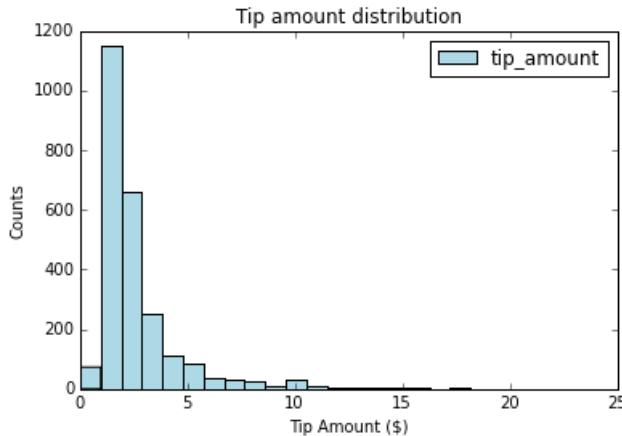
# PLOT TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

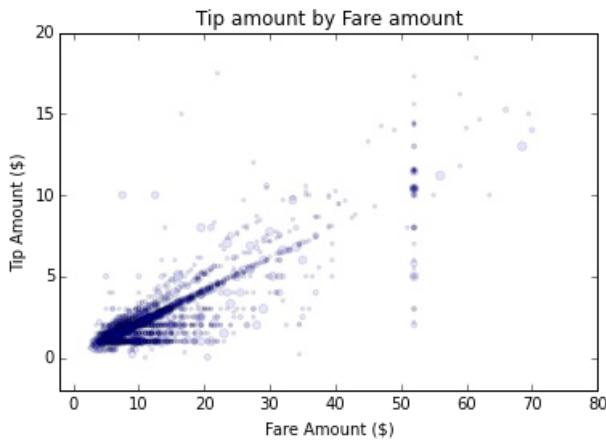
# PLOT TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# PLOT TIP AMOUNT BY FARE AMOUNT; SCALE POINTS BY PASSENGER COUNT
ax = sqlResults.plot(kind='scatter', x= 'fare_amount', y = 'tip_amount', c='blue', alpha = 0.10, s=5*
(sqlResults.passenger_count))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.show()

```

Output:





Create features and transform features, and then prep data for input into modeling functions

For tree-based modeling functions from Spark ML and MLLib, you have to prepare target and features by using a variety of techniques, such as binning, indexing, one-hot encoding, and vectorization. Here are the procedures to follow in this section:

1. Create a new feature by **binning** hours into traffic time buckets.
2. Apply **indexing** and **one-hot encoding** to categorical features.
3. **Sample and split the data set** into training and test fractions.
4. **Specify training variable and features**, and then create indexed or one-hot encoded training and testing input labeled point resilient distributed datasets (RDDs) or data frames.
5. Automatically **categorize and vectorize features and targets** to use as inputs for machine learning models.

Create a new feature by binning hours into traffic time buckets

This code shows you how to create a new feature by binning hours into traffic time buckets and how to cache the resulting data frame in memory. Where RDDs and data frames are used repeatedly, caching leads to improved execution times. Accordingly, you'll cache RDDs and data frames at several stages in the following procedures.

```
# CREATE FOUR BUCKETS FOR TRAFFIC TIMES
val sqlStatement = """
    SELECT *,
    CASE
        WHEN (pickup_hour <= 6 OR pickup_hour >= 20) THEN "Night"
        WHEN (pickup_hour >= 7 AND pickup_hour <= 10) THEN "AMRush"
        WHEN (pickup_hour >= 11 AND pickup_hour <= 15) THEN "Afternoon"
        WHEN (pickup_hour >= 16 AND pickup_hour <= 19) THEN "PMRush"
    END as TrafficTimeBins
    FROM taxi_train
"""

val taxi_df_train_with_newFeatures = sqlContext.sql(sqlStatement)

# CACHE THE DATA FRAME IN MEMORY AND MATERIALIZE THE DATA FRAME IN MEMORY
taxi_df_train_with_newFeatures.cache()
taxi_df_train_with_newFeatures.count()
```

Indexing and one-hot encoding of categorical features

The modeling and predict functions of MLLib require features with categorical input data to be indexed or encoded prior to use. This section shows you how to index or encode categorical features for input into the modeling functions.

You need to index or encode your models in different ways, depending on the model. For example, logistic and linear regression models require one-hot encoding. For example, a feature with three categories can be expanded into three feature columns. Each column would contain 0 or 1 depending on the category of an observation. MLlib provides the [OneHotEncoder](#) function for one-hot encoding. This encoder maps a column of label indices to a column of binary vectors with at most a single one-value. With this encoding, algorithms that expect numerical valued features, such as logistic regression, can be applied to categorical features.

Here you transform only four variables to show examples, which are character strings. You also can index other variables, such as weekday, represented by numerical values, as categorical variables.

For indexing, use `StringIndexer()`, and for one-hot encoding, use `OneHotEncoder()` functions from MLlib. Here is the code to index and encode categorical features:

```
# CREATE INDEXES AND ONE-HOT ENCODED VECTORS FOR SEVERAL CATEGORICAL FEATURES

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# INDEX AND ENCODE VENDOR_ID
val stringIndexer = new
StringIndexer().setInputCol("vendor_id").setOutputCol("vendorIndex").fit(taxi_df_train_with_newFeatures)
val indexed = stringIndexer.transform(taxi_df_train_with_newFeatures)
val encoder = new OneHotEncoder().setInputCol("vendorIndex").setOutputCol("vendorVec")
val encoded1 = encoder.transform(indexed)

# INDEX AND ENCODE RATE_CODE
val stringIndexer = new StringIndexer().setInputCol("rate_code").setOutputCol("rateIndex").fit(encoded1)
val indexed = stringIndexer.transform(encoded1)
val encoder = new OneHotEncoder().setInputCol("rateIndex").setOutputCol("rateVec")
val encoded2 = encoder.transform(indexed)

# INDEX AND ENCODE PAYMENT_TYPE
val stringIndexer = new
StringIndexer().setInputCol("payment_type").setOutputCol("paymentIndex").fit(encoded2)
val indexed = stringIndexer.transform(encoded2)
val encoder = new OneHotEncoder().setInputCol("paymentIndex").setOutputCol("paymentVec")
val encoded3 = encoder.transform(indexed)

# INDEX AND TRAFFIC TIME BINS
val stringIndexer = new
StringIndexer().setInputCol("TrafficTimeBins").setOutputCol("TrafficTimeBinsIndex").fit(encoded3)
val indexed = stringIndexer.transform(encoded3)
val encoder = new OneHotEncoder().setInputCol("TrafficTimeBinsIndex").setOutputCol("TrafficTimeBinsVec")
val encodedFinal = encoder.transform(indexed)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");
```

Output:

Time to run the cell: 4 seconds.

Sample and split the data set into training and test fractions

This code creates a random sampling of the data (25%, in this example). Although sampling is not required for this example due to the size of the data set, the article shows you how you can sample so that you know how to use it for your own problems when needed. When samples are large, this can save significant time while you train models. Next, split the sample into a training part (75%, in this example) and a testing part (25%, in this example) to use in classification and regression modeling.

Add a random number (between 0 and 1) to each row (in a "rand" column) that can be used to select cross-

validation folds during training.

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
val samplingFraction = 0.25;
val trainingFraction = 0.75;
val testingFraction = (1-trainingFraction);
val seed = 1234;
val encodedFinalSampledTmp = encodedFinal.sample(withReplacement = false, fraction = samplingFraction, seed = seed)
val sampledDFcount = encodedFinalSampledTmp.count().toInt

val generateRandomDouble = udf(() => {
    scala.util.Random.nextDouble
})

# ADD A RANDOM NUMBER FOR CROSS-VALIDATION
val encodedFinalSampled = encodedFinalSampledTmp.withColumn("rand", generateRandomDouble());

# SPLIT THE SAMPLED DATA FRAME INTO TRAIN AND TEST, WITH A RANDOM COLUMN ADDED FOR DOING CROSS-VALIDATION (SHOWN LATER)
# INCLUDE A RANDOM COLUMN FOR CREATING CROSS-VALIDATION FOLDS
val splits = encodedFinalSampled.randomSplit(Array(trainingFraction, testingFraction), seed = seed)
val trainData = splits(0)
val testData = splits(1)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");
```

Output:

Time to run the cell: 2 seconds.

Specify training variable and features, and then create indexed or one-hot encoded training and testing input labeled point RDDs or data frames

This section contains code that shows you how to index categorical text data as a labeled point data type, and encode it so you can use it to train and test MLlib logistic regression and other classification models. Labeled point objects are RDDs that are formatted in a way that is needed as input data by most of machine learning algorithms in MLlib. A [labeled point](#) is a local vector, either dense or sparse, associated with a label/response.

In this code, you specify the target (dependent) variable and the features to use to train models. Then, you create indexed or one-hot encoded training and testing input labeled point RDDs or data frames.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# MAP NAMES OF FEATURES AND TARGETS FOR CLASSIFICATION AND REGRESSION PROBLEMS
val featuresIndOneHot = List("paymentVec", "vendorVec", "rateVec", "TrafficTimeBinsVec", "pickup_hour",
"weekday", "passenger_count", "trip_time_in_secs", "trip_distance",
"fare_amount").map(encodedFinalSampled.columns.indexOf(_))
val featuresIndIndex = List("paymentIndex", "vendorIndex", "rateIndex", "TrafficTimeBinsIndex",
"pickup_hour", "weekday", "passenger_count", "trip_time_in_secs", "trip_distance",
"fare_amount").map(encodedFinalSampled.columns.indexOf(_))

# SPECIFY THE TARGET FOR CLASSIFICATION ('tipped') AND REGRESSION ('tip_amount') PROBLEMS
val targetIndBinary = List("tipped").map(encodedFinalSampled.columns.indexOf(_))
val targetIndRegression = List("tip_amount").map(encodedFinalSampled.columns.indexOf(_))

# CREATE INDEXED LABELED POINT RDD OBJECTS
val indexedTRAINbinary = trainData.rdd.map(r => LabeledPoint(r.getDouble(targetIndBinary(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)))
val indexedTESTbinary = testData.rdd.map(r => LabeledPoint(r.getDouble(targetIndBinary(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)))
val indexedTRAINreg = trainData.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)))
val indexedTESTreg = testData.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)))

# CREATE INDEXED DATA FRAMES THAT YOU CAN USE TO TRAIN BY USING SPARK ML FUNCTIONS
val indexedTRAINbinaryDF = indexedTRAINbinary.toDF()
val indexedTESTbinaryDF = indexedTESTbinary.toDF()
val indexedTRAINregDF = indexedTRAINreg.toDF()
val indexedTESTregDF = indexedTESTreg.toDF()

# CREATE ONE-HOT ENCODED (VECTORIZED) DATA FRAMES THAT YOU CAN USE TO TRAIN BY USING SPARK ML FUNCTIONS
val assemblerOneHot = new VectorAssembler().setInputCols(Array("paymentVec", "vendorVec", "rateVec",
"TrafficTimeBinsVec", "pickup_hour", "weekday", "passenger_count", "trip_time_in_secs", "trip_distance",
"fare_amount")).setOutputCol("features")
val OneHotTRAIN = assemblerOneHot.transform(trainData)
val OneHotTEST = assemblerOneHot.transform(testData)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 4 seconds.

Automatically categorize and vectorize features and targets to use as inputs for machine learning models

Use Spark ML to categorize the target and features to use in tree-based modeling functions. The code completes two tasks:

- Creates a binary target for classification by assigning a value of 0 or 1 to each data point between 0 and 1 by using a threshold value of 0.5.
- Automatically categorizes features. If the number of distinct numerical values for any feature is less than 32, that feature is categorized.

Here's the code for these two tasks.

```

# CATEGORIZE FEATURES AND BINARIZE THE TARGET FOR THE BINARY CLASSIFICATION PROBLEM

# TRAIN DATA
val indexer = new VectorIndexer().setInputCol("features").setOutputCol("featuresCat").setMaxCategories(32)
val indexerModel = indexer.fit(indexedTRAINbinaryDF)
val indexedTrainwithCatFeat = indexerModel.transform(indexedTRAINbinaryDF)
val binarizer: Binarizer = new Binarizer().setInputCol("label").setOutputCol("labelBin").setThreshold(0.5)
val indexedTRAINwithCatFeatBinTarget = binarizer.transform(indexedTrainwithCatFeat)

# TEST DATA
val indexerModel = indexer.fit(indexedTESTbinaryDF)
val indexedTrainwithCatFeat = indexerModel.transform(indexedTESTbinaryDF)
val binarizer: Binarizer = new Binarizer().setInputCol("label").setOutputCol("labelBin").setThreshold(0.5)
val indexedTESTwithCatFeatBinTarget = binarizer.transform(indexedTrainwithCatFeat)

# CATEGORIZE FEATURES FOR THE REGRESSION PROBLEM
# CREATE PROPERLY INDEXED AND CATEGORIZED DATA FRAMES FOR TREE-BASED MODELS

# TRAIN DATA
val indexer = new VectorIndexer().setInputCol("features").setOutputCol("featuresCat").setMaxCategories(32)
val indexerModel = indexer.fit(indexedTRAINregDF)
val indexedTRAINwithCatFeat = indexerModel.transform(indexedTRAINregDF)

# TEST DATA
val indexerModel = indexer.fit(indexedTESTbinaryDF)
val indexedTESTwithCatFeat = indexerModel.transform(indexedTESTregDF)

```

Binary classification model: Predict whether a tip should be paid

In this section, you create three types of binary classification models to predict whether or not a tip should be paid:

- A **logistic regression model** by using the Spark ML `LogisticRegression()` function
- A **random forest classification model** by using the Spark ML `RandomForestClassifier()` function
- A **gradient boosting tree classification model** by using the MLlib `GradientBoostedTrees()` function

Create a logistic regression model

Next, create a logistic regression model by using the Spark ML `LogisticRegression()` function. You create the model building code in a series of steps:

1. **Train the model** data with one parameter set.
2. **Evaluate the model** on a test data set with metrics.
3. **Save the model** in Blob storage for future consumption.
4. **Score the model** against test data.
5. **Plot the results** with receiver operating characteristic (ROC) curves.

Here's the code for these procedures:

```

# CREATE A LOGISTIC REGRESSION MODEL
val lr = new LogisticRegression().setLabelCol("tipped").setFeaturesCol("features").setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(OneHotTRAIN)

# PREDICT ON THE TEST DATA SET
val predictions = lrModel.transform(OneHotTEST)

# SELECT `BinaryClassificationEvaluator()` TO COMPUTE THE TEST ERROR
val evaluator = new BinaryClassificationEvaluator().setLabelCol("tipped").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)
println("ROC on test data = " + ROC)

# SAVE THE MODEL
val timestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LogisticRegression_"
val filename = modelDir.concat(modelName).concat(timestamp)
lrModel.save(filename);

```

Load, score, and save the results.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# LOAD THE SAVED MODEL AND SCORE THE TEST DATA SET
val savedModel = org.apache.spark.ml.classification.LogisticRegressionModel.load(filename)
println(s"Coefficients: ${savedModel.coefficients} Intercept: ${savedModel.intercept}")

# SCORE THE MODEL ON THE TEST DATA
val predictions = savedModel.transform(OneHotTEST).select("tipped", "probability", "rawPrediction")
predictions.registerTempTable("testResults")

# SELECT `BinaryClassificationEvaluator()` TO COMPUTE THE TEST ERROR
val evaluator = new BinaryClassificationEvaluator().setLabelCol("tipped").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

# PRINT THE ROC RESULTS
println("ROC on test data = " + ROC)

```

Output:

ROC on test data = 0.9827381497557599

Use Python on local Pandas data frames to plot the ROC curve.

```

# QUERY THE RESULTS
%%sql -q -o sqlResults
SELECT tipped, probability from testResults

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline
from sklearn.metrics import roc_curve,auc

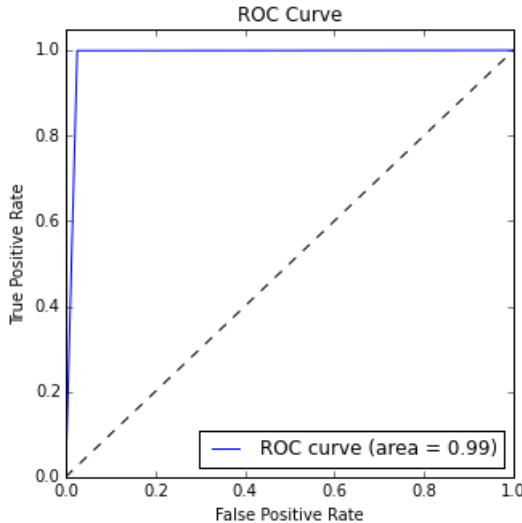
sqlResults['probFloat'] = sqlResults.apply(lambda row: row['probability'].values()[0][1], axis=1)
predictions_pddf = sqlResults[['tipped',"probFloat"]]

# PREDICT THE ROC CURVE
# predictions_pddf = sqlResults.rename(columns={'_1': 'probability', 'tipped': 'label'})
prob = predictions_pddf["probFloat"]
fpr, tpr, thresholds = roc_curve(predictions_pddf['tipped'], prob, pos_label=1);
roc_auc = auc(fpr, tpr)

# PLOT THE ROC CURVE
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

```

Output:



Create a random forest classification model

Next, create a random forest classification model by using the Spark ML `RandomForestClassifier()` function, and then evaluate the model on test data.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE THE RANDOM FOREST CLASSIFIER MODEL
val rf = new
RandomForestClassifier().setLabelCol("labelBin").setFeaturesCol("featuresCat").setNumTrees(10).setSeed(1234)

# FIT THE MODEL
val rfModel = rf.fit(indexedTRAINwithCatFeatBinTarget)
val predictions = rfModel.transform(indexedTESTwithCatFeatBinTarget)

# EVALUATE THE MODEL
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("f1")
val Test_f1Score = evaluator.evaluate(predictions)
println("F1 score on test data: " + Test_f1Score);

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# CALCULATE BINARY CLASSIFICATION EVALUATION METRICS
val evaluator = new
BinaryClassificationEvaluator().setLabelCol("label").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)
println("ROC on test data = " + ROC)

```

Output:

ROC on test data = 0.9847103571552683

Create a GBT classification model

Next, create a GBT classification model by using MLlib's `GradientBoostedTrees()` function, and then evaluate the model on test data.

```

# TRAIN A GBT CLASSIFICATION MODEL BY USING MLLIB AND A LABELED POINT

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE GBT CLASSIFICATION MODEL
val boostingStrategy = BoostingStrategy.defaultParams("Classification")
boostingStrategy.numIterations = 20
boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 5
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]((0,2),(1,2),(2,6),(3,4))

# TRAIN THE MODEL
val gbtModel = GradientBoostedTrees.train(indexedTRAINbinary, boostingStrategy)

# SAVE THE MODEL IN BLOB STORAGE
val timestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "GBT_Classification_"
val filename = modelDir.concat(modelName).concat(timestamp)
gbtModel.save(sc, filename);

# EVALUATE THE MODEL ON TEST INSTANCES AND THE COMPUTE TEST ERROR
val labelAndPreds = indexedTESTbinary.map { point =>
    val prediction = gbtModel.predict(point.features)
    (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / indexedTRAINbinary.count()
//println("Learned classification GBT model:\n" + gbtModel.toDebugString)
println("Test Error = " + testErr)

# USE BINARY AND MULTICLASS METRICS TO EVALUATE THE MODEL ON THE TEST DATA
val metrics = new MulticlassMetrics(labelAndPreds)
println(s"Precision: ${metrics.precision}")
println(s"Recall: ${metrics.recall}")
println(s"F1 Score: ${metrics.fMeasure}")

val metrics = new BinaryClassificationMetrics(labelAndPreds)
println(s"Area under PR curve: ${metrics.areaUnderPR}")
println(s"Area under ROC curve: ${metrics.areaUnderROC}")

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# PRINT THE ROC METRIC
println(s"Area under ROC curve: ${metrics.areaUnderROC}")

```

Output:

Area under ROC curve: 0.9846895479241554

Regression model: Predict tip amount

In this section, you create two types of regression models to predict the tip amount:

- A **regularized linear regression model** by using the Spark ML `LinearRegression()` function. You'll save the model and evaluate the model on test data.
- A **gradient-boosting tree regression model** by using the Spark ML `GBTRegressor()` function.

Create a regularized linear regression model

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE A REGULARIZED LINEAR REGRESSION MODEL BY USING THE SPARK ML FUNCTION AND DATA FRAMES
val lr = new
LinearRegression().setLabelCol("tip_amount").setFeaturesCol("features").setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

# FIT THE MODEL BY USING DATA FRAMES
val lrModel = lr.fit(OneHotTRAIN)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

# SUMMARIZE THE MODEL OVER THE TRAINING SET AND PRINT METRICS
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")

# SAVE THE MODEL IN AZURE BLOB STORAGE
val datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LinearRegression_"
val filename = modelDir.concat(modelName).concat(datestamp)
lrModel.save(filename);

# PRINT THE COEFFICIENTS
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

# SCORE THE MODEL ON TEST DATA
val predictions = lrModel.transform(OneHotTEST)

# EVALUATE THE MODEL ON TEST DATA
val evaluator = new
RegressionEvaluator().setLabelCol("tip_amount").setPredictionCol("prediction").setMetricName("r2")
val r2 = evaluator.evaluate(predictions)
println("R-sqr on test data = " + r2)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 13 seconds.

```

# LOAD A SAVED LINEAR REGRESSION MODEL FROM BLOB STORAGE AND SCORE A TEST DATA SET

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# LOAD A SAVED LINEAR REGRESSION MODEL FROM AZURE BLOB STORAGE
val savedModel = org.apache.spark.ml.regression.LinearRegressionModel.load(filename)
println(s"Coefficients: ${savedModel.coefficients} Intercept: ${savedModel.intercept}")

# SCORE THE MODEL ON TEST DATA
val predictions = savedModel.transform(OneHotTEST).select("tip_amount","prediction")
predictions.registerTempTable("testResults")

# EVALUATE THE MODEL ON TEST DATA
val evaluator = new
RegressionEvaluator().setLabelCol("tip_amount").setPredictionCol("prediction").setMetricName("r2")
val r2 = evaluator.evaluate(predictions)
println("R-sqr on test data = " + r2)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

# PRINT THE RESULTS
println("R-sqr on test data = " + r2)

```

Output:

R-sqr on test data = 0.5960320470835743

Next, query the test results as a data frame and use AutoVizWidget and matplotlib to visualize it.

```

# RUN A SQL QUERY
%%sql -q -o sqlResults
select * from testResults

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES
# CLICK THE TYPE OF PLOT TO GENERATE (LINE, AREA, BAR, AND SO ON)
sqlResults

```

The code creates a local data frame from the query output and plots the data. The `%%local` magic creates a local data frame, `sqlResults`, which you can use to plot with matplotlib.

NOTE

This Spark magic is used multiple times in this article. If the amount of data is large, you should sample to create a data frame that can fit in local memory.

Create plots by using Python matplotlib.

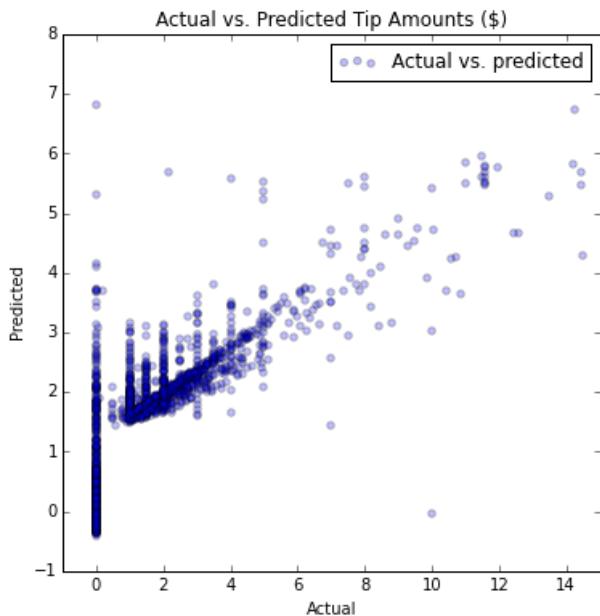
```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
sqlResults
%matplotlib inline
import numpy as np

# PLOT THE RESULTS
ax = sqlResults.plot(kind='scatter', figsize = (6,6), x='tip_amount', y='prediction', color='blue', alpha = 0.25, label='Actual vs. predicted');
fit = np.polyfit(sqlResults['tip_amount'], sqlResults['prediction'], deg=1)
ax.set_title('Actual vs. Predicted Tip Amounts ($)')
ax.set_xlabel("Actual")
ax.set_ylabel("Predicted")
#ax.plot(sqlResults['tip_amount'], fit[0] * sqlResults['prediction'] + fit[1], color='magenta')
plt.axis([-1, 15, -1, 8])
plt.show(ax)

```

Output:



Create a GBT regression model

Create a GBT regression model by using the Spark ML `GBTRegressor()` function, and then evaluate the model on test data.

[Gradient-boosted trees](#) (GBTS) are ensembles of decision trees. GBTS trains decision trees iteratively to minimize a loss function. You can use GBTS for regression and classification. They can handle categorical features, do not require feature scaling, and can capture nonlinearities and feature interactions. You also can use them in a multiclass-classification setting.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# TRAIN A GBT REGRESSION MODEL
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("featuresCat").setMaxIter(10)
val gbtModel = gbt.fit(indexedTRAINwithCatFeat)

# MAKE PREDICTIONS
val predictions = gbtModel.transform(indexedTESTwithCatFeat)

# COMPUTE TEST SET R2
val evaluator = new
RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(predictions)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

# PRINT THE RESULTS
println("Test R-sqr is: " + Test_R2);

```

Output:

Test R-sqr is: 0.7655383534596654

Advanced modeling utilities for optimization

In this section, you use machine learning utilities that developers frequently use for model optimization. Specifically, you can optimize machine learning models three different ways by using parameter sweeping and cross-validation:

- Split the data into train and validation sets, optimize the model by using hyper-parameter sweeping on a training set, and evaluate on a validation set (linear regression)
- Optimize the model by using cross-validation and hyper-parameter sweeping by using Spark ML's CrossValidator function (binary classification)
- Optimize the model by using custom cross-validation and parameter-sweeping code to use any machine learning function and parameter set (linear regression)

Cross-validation is a technique that assesses how well a model trained on a known set of data will generalize to predict the features of data sets on which it has not been trained. The general idea behind this technique is that a model is trained on a data set of known data, and then the accuracy of its predictions is tested against an independent data set. A common implementation is to divide a data set into k -folds, and then train the model in a round-robin fashion on all but one of the folds.

Hyper-parameter optimization is the problem of choosing a set of hyper-parameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. A hyper-parameter is a value that you must specify outside the model training procedure. Assumptions about hyper-parameter values can affect the flexibility and accuracy of the model. Decision trees have hyper-parameters, for example, such as the desired depth and number of leaves in the tree. You must set a misclassification penalty term for a support vector machine (SVM).

A common way to perform hyper-parameter optimization is to use a grid search, also called a **parameter sweep**. In a grid search, an exhaustive search is performed through the values of a specified subset of the hyper-parameter space for a learning algorithm. Cross-validation can supply a performance metric to sort out the optimal results produced by the grid search algorithm. If you use cross-validation hyper-parameter sweeping, you can help limit problems like overfitting a model to training data. This way, the model retains the

capacity to apply to the general set of data from which the training data was extracted.

Optimize a linear regression model with hyper-parameter sweeping

Next, split data into train and validation sets, use hyper-parameter sweeping on a training set to optimize the model, and evaluate on a validation set (linear regression).

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# RENAME `tip_amount` AS A LABEL
val OneHotTRAINLabeled =
  OneHotTRAIN.select("tip_amount", "features").withColumnRenamed(existingName="tip_amount",newName="label")
val OneHotTESTLabeled =
  OneHotTEST.select("tip_amount", "features").withColumnRenamed(existingName="tip_amount",newName="label")
OneHotTRAINLabeled.cache()
OneHotTESTLabeled.cache()

# DEFINE THE ESTIMATOR FUNCTION: `THE LinearRegression()` FUNCTION
val lr = new LinearRegression().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)

# DEFINE THE PARAMETER GRID
val paramGrid = new ParamGridBuilder().addGrid(lr.regParam, Array(0.1, 0.01,
  0.001)).addGrid(lr.fitIntercept).addGrid(lr.elasticNetParam, Array(0.1, 0.5, 0.9)).build()

# DEFINE THE PIPELINE WITH A TRAIN/TEST VALIDATION SPLIT (75% IN THE TRAINING SET), AND THEN THE SPECIFY
ESTIMATOR, EVALUATOR, AND PARAMETER GRID
val trainPct = 0.75
val trainValidationSplit = new TrainValidationSplit().setEstimator(lr).setEvaluator(new
RegressionEvaluator).setEstimatorParamMaps(paramGrid).setTrainRatio(trainPct)

# RUN THE TRAIN VALIDATION SPLIT AND CHOOSE THE BEST SET OF PARAMETERS
val model = trainValidationSplit.fit(OneHotTRAINLabeled)

# MAKE PREDICTIONS ON THE TEST DATA BY USING THE MODEL WITH THE COMBINATION OF PARAMETERS THAT PERFORMS THE
BEST
val testResults = model.transform(OneHotTESTLabeled).select("label", "prediction")

# COMPUTE TEST SET R2
val evaluator = new
RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(testResults)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

println("Test R-sqr is: " + Test_R2);
```

Output:

Test R-sqr is: 0.6226484708501209

Optimize the binary classification model by using cross-validation and hyper-parameter sweeping

This section shows you how to optimize a binary classification model by using cross-validation and hyper-parameter sweeping. This uses the Spark ML `CrossValidator` function.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE DATA FRAMES WITH PROPERLY LABELED COLUMNS TO USE WITH THE TRAIN AND TEST SPLIT
val indexedTRAINwithCatFeatBinTargetRF =
indexedTRAINwithCatFeatBinTarget.select("labelBin","featuresCat").withColumnRenamed(existingName="labelBin",
newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
val indexedTESTwithCatFeatBinTargetRF =
indexedTESTwithCatFeatBinTarget.select("labelBin","featuresCat").withColumnRenamed(existingName="labelBin",
newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
indexedTRAINwithCatFeatBinTargetRF.cache()
indexedTESTwithCatFeatBinTargetRF.cache()

# DEFINE THE ESTIMATOR FUNCTION
val rf = new
RandomForestClassifier().setLabelCol("label").setFeaturesCol("features").setImpurity("gini").setSeed(1234).s
etFeatureSubsetStrategy("auto").setMaxBins(32)

# DEFINE THE PARAMETER GRID
val paramGrid = new ParamGridBuilder().addGrid(rf.maxDepth, Array(4,8)).addGrid(rf.numTrees,
Array(5,10)).addGrid(rf.minInstancesPerNode, Array(100,300)).build()

# SPECIFY THE NUMBER OF FOLDS
val numFolds = 3

# DEFINE THE TRAIN/TEST VALIDATION SPLIT (75% IN THE TRAINING SET)
val CrossValidator = new CrossValidator().setEstimator(rf).setEvaluator(new
BinaryClassificationEvaluator).setEstimatorParamMaps(paramGrid).setNumFolds(numFolds)

# RUN THE TRAIN VALIDATION SPLIT AND CHOOSE THE BEST SET OF PARAMETERS
val model = CrossValidator.fit(indexedTRAINwithCatFeatBinTargetRF)

# MAKE PREDICTIONS ON THE TEST DATA BY USING THE MODEL WITH THE COMBINATION OF PARAMETERS THAT PERFORMS THE
BEST
val testResults = model.transform(indexedTESTwithCatFeatBinTargetRF).select("label", "prediction")

# COMPUTE THE TEST F1 SCORE
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("f1")
val Test_f1Score = evaluator.evaluate(testResults)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 33 seconds.

Optimize the linear regression model by using custom cross-validation and parameter-sweeping code

Next, optimize the model by using custom code, and identify the best model parameters by using the criterion of highest accuracy. Then, create the final model, evaluate the model on test data, and save the model in Blob storage. Finally, load the model, score test data, and evaluate accuracy.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE PARAMETER GRID AND THE NUMBER OF FOLDS
val paramGrid = new ParamGridBuilder().addGrid(rf.maxDepth, Array(5,10)).addGrid(rf.numTrees,
Array(10,25,50)).build()

val nFolds = 3
val numModels = paramGrid.size
val numParamsinGrid = 2

```

```

# SPECIFY THE NUMBER OF CATEGORIES FOR CATEGORICAL VARIABLES
val categoricalFeaturesInfo = Map[Int, Int]((0,2),(1,2),(2,6),(3,4))

var maxDepth = -1
var numTrees = -1
var param = ""
var paramval = -1
var validateLB = -1.0
var validateUB = -1.0
val h = 1.0 / nFolds;
val RMSE = Array.fill(numModels)(0.0)

# CREATE K-FOLDS
val splits = MLUtils.kFold(indexedTRAINbinary, numFolds = nFolds, seed=1234)

# LOOP THROUGH K-FOLDS AND THE PARAMETER GRID TO GET AND IDENTIFY THE BEST PARAMETER SET BY LEVEL OF
ACCURACY
for (i <- 0 to (nFolds-1)) {
    validateLB = i * h
    validateUB = (i + 1) * h
    val validationCV = trainData.filter($"rand" >= validateLB && $"rand" < validateUB)
    val trainCV = trainData.filter($"rand" < validateLB || $"rand" >= validateUB)
    val validationLabPt = validationCV.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)));
    val trainCVLabPt = trainCV.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_)).toArray)));
    validationLabPt.cache()
    trainCVLabPt.cache()

    for (nParamSets <- 0 to (numModels-1)) {
        for (nParams <- 0 to (numParamsinGrid-1)) {
            param = paramGrid(nParamSets).toSeq(nParams).param.toString.split("__")(1)
            paramval = paramGrid(nParamSets).toSeq(nParams).value.toString.toInt
            if (param == "maxDepth") {maxDepth = paramval}
            if (param == "numTrees") {numTrees = paramval}
        }
        val rfModel = RandomForest.trainRegressor(trainCVLabPt,
categoricalFeaturesInfo=categoricalFeaturesInfo,
                                              numTrees=numTrees, maxDepth=maxDepth,
                                              featureSubsetStrategy="auto", impurity="variance",
maxBins=32)
        val labelAndPreds = validationLabPt.map { point =>
                                              val prediction = rfModel.predict(point.features)
                                              ( prediction, point.label )
                                         }
        val validMetrics = new RegressionMetrics(labelAndPreds)
        val rmse = validMetrics.rootMeanSquaredError
        RMSE(nParamSets) += rmse
    }
    validationLabPt.unpersist();
    trainCVLabPt.unpersist();
}
val minRMSEindex = RMSE.indexOf(RMSE.min)

# GET THE BEST PARAMETERS FROM A CROSS-VALIDATION AND PARAMETER SWEEP
var best_maxDepth = -1
var best_numTrees = -1
for (nParams <- 0 to (numParamsinGrid-1)) {
    param = paramGrid(minRMSEindex).toSeq(nParams).param.toString.split("__")(1)
    paramval = paramGrid(minRMSEindex).toSeq(nParams).value.toString.toInt
    if (param == "maxDepth") {best_maxDepth = paramval}
    if (param == "numTrees") {best_numTrees = paramval}
}

# CREATE THE BEST MODEL WITH THE BEST PARAMETERS AND A FULL TRAINING DATA SET
val best_rfModel = RandomForest.trainRegressor(indexedTRAINreg,
categoricalFeaturesInfo=categoricalFeaturesInfo,
                                              numTrees=best_numTrees, maxDepth=best_maxDepth

```

```

    numTrees=bestNumTrees, maxDepth=bestMaxDepth,
    featureSubsetStrategy="auto", impurity="variance",
    maxBins=32)

# SAVE THE BEST RANDOM FOREST MODEL IN BLOB STORAGE
val datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "BestCV_RF_Regression_"
val filename = modelDir.concat(modelName).concat(datestamp)
best_rfModel.save(sc, filename);

# PREDICT ON THE TRAINING SET WITH THE BEST MODEL AND THEN EVALUATE
val labelAndPreds = indexedTESTreg.map { point =>
    val prediction = best_rfModel.predict(point.features)
    ( prediction, point.label )
}

val test_rmse = new RegressionMetrics(labelAndPreds).rootMeanSquaredError
val test_rsqr = new RegressionMetrics(labelAndPreds).r2

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# LOAD THE MODEL
val savedRFModel = RandomForestModel.load(sc, filename)

val labelAndPreds = indexedTESTreg.map { point =>
    val prediction = savedRFModel.predict(point.features)
    ( prediction, point.label )
}

# TEST THE MODEL
val test_rmse = new RegressionMetrics(labelAndPreds).rootMeanSquaredError
val test_rsqr = new RegressionMetrics(labelAndPreds).r2

```

Output:

Time to run the cell: 61 seconds.

Next steps

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Feature engineering in machine learning

3/10/2022 • 6 minutes to read • [Edit Online](#)

NOTE

This item is under maintenance. We encourage you to use the [Azure Machine Learning designer](#).

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to [Azure Machine Learning](#) by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources. Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) resources.

- See [information on moving machine learning projects from ML Studio \(classic\) to Azure Machine Learning](#).
- Learn more about [Azure Machine Learning](#)

ML Studio (classic) documentation is being retired and may not be updated in the future.

In this article, you learn about feature engineering and its role in enhancing data in machine learning. Learn from illustrative examples drawn from [Azure Machine Learning Studio \(classic\)](#) experiments.

- **Feature engineering:** The process of creating new features from raw data to increase the predictive power of the learning algorithm. Engineered features should capture additional information that is not easily apparent in the original feature set.
- **Feature selection:** The process of selecting the key subset of features to reduce the dimensionality of the training problem.

Normally **feature engineering** is applied first to generate additional features, and then **feature selection** is done to eliminate irrelevant, redundant, or highly correlated features.

Feature engineering and selection are part of the [modeling stage](#) of the Team Data Science Process (TDSP). To learn more about the TDSP and the data science lifecycle, see [What is the TDSP?](#)

What is feature engineering?

Training data consists of a matrix composed of rows and columns. Each row in the matrix is an observation or record. The columns of each row are the features that describe each record. The features specified in the experimental design should characterize the patterns in the data.

Although many of the raw data fields can be used directly to train a model, it's often necessary to create additional (engineered) features for an enhanced training dataset.

Engineered features that enhance training provide information that better differentiates the patterns in the data. But this process is something of an art. Sound and productive decisions often require domain expertise.

Example 1: Add temporal features for a regression model

Let's use the experiment [Demand forecasting of bikes rentals](#) in Azure Machine Learning Studio (classic) to demonstrate how to engineer features for a regression task. The objective of this experiment is to predict the

demand for bike rentals within a specific month/day/hour.

Bike rental dataset

The [Bike Rental UCI dataset](#) is based on real data from a bike share company based in the United States. It represents the number of bike rentals within a specific hour of a day for the years 2011 and 2012. It contains 17,379 rows and 17 columns.

The raw feature set contains weather conditions (temperature/humidity/wind speed) and the type of the day (holiday/weekday). The field to predict is the count, which represents the bike rentals within a specific hour. Count ranges from 1 to 977.

Create a feature engineering experiment

With the goal of constructing effective features in the training data, four regression models are built using the same algorithm but with four different training datasets. The four datasets represent the same raw input data, but with an increasing number of features set. These features are grouped into four categories:

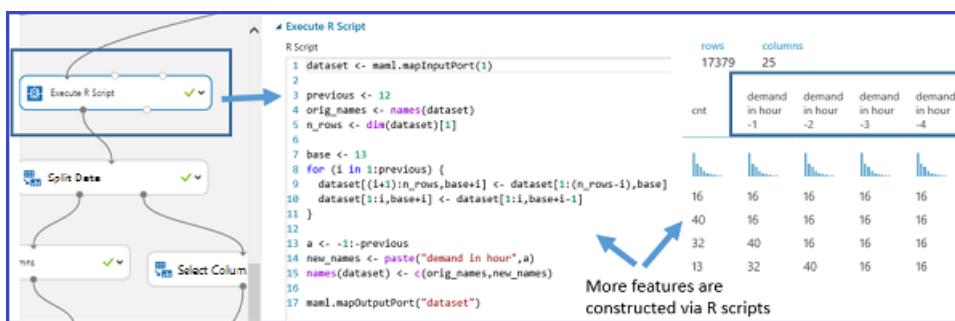
1. A = weather + holiday + weekday + weekend features for the predicted day
2. B = number of bikes that were rented in each of the previous 12 hours
3. C = number of bikes that were rented in each of the previous 12 days at the same hour
4. D = number of bikes that were rented in each of the previous 12 weeks at the same hour and the same day

Besides feature set A, which already exists in the original raw data, the other three sets of features are created through the feature engineering process. Feature set B captures recent demand for the bikes. Feature set C captures the demand for bikes at a particular hour. Feature set D captures demand for bikes at particular hour and particular day of the week. The four training datasets each includes feature set A, A+B, A+B+C, and A+B+C+D, respectively.

Feature engineering using Studio (classic)

In the Studio (classic) experiment, these four training datasets are formed via four branches from the pre-processed input dataset. Except for the leftmost branch, each of these branches contains an [Execute R Script](#) module, in which the derived features (feature set B, C, and D) are constructed and appended to the imported dataset.

The following figure demonstrates the R script used to create feature set B in the second left branch.



Results

A comparison of the performance results of the four models is summarized in the following table:

Features	Mean Absolute Error	Root Mean Square Error
A	89.7	124.9
A + B	51.7	88.3
A + B + C	47.6	81.1
A + B + C + D	48.3	82.1

The best results are shown by features A+B+C. The error rate decreases when additional feature set are included in the training data. It verifies the presumption that the feature set B, C provide additional relevant information

for the regression task. But adding the D feature does not seem to provide any additional reduction in the error rate.

Example 2: Create features for text mining

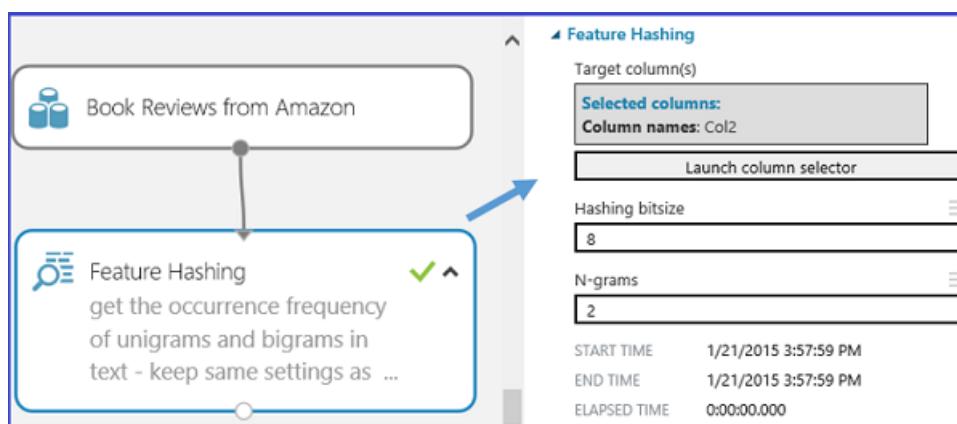
Feature engineering is widely applied in tasks related to text mining such as document classification and sentiment analysis. Since individual pieces of raw text usually serve as the input data, the feature engineering process is needed to create the features involving word/phrase frequencies.

Feature hashing

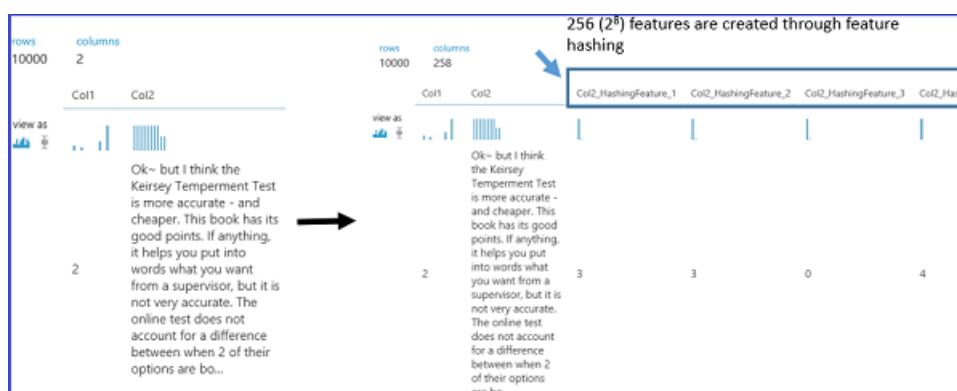
To achieve this task, a technique called **feature hashing** is applied to efficiently turn arbitrary text features into indices. Instead of associating each text feature (words/phrases) to a particular index, this method applies a hash function to the features and using their hash values as indices directly.

In Studio (classic), there is a **Feature Hashing** module that creates word/phrase features conveniently. Following figure shows an example of using this module. The input dataset contains two columns: the book rating ranging from 1 to 5, and the actual review content. The goal of this module is to retrieve a bunch of new features that show the occurrence frequency of the corresponding word(s)/phrase(s) within the particular book review. To use this module, complete the following steps:

- First, select the column that contains the input text ("Col2" in this example).
- Second, set the "Hashing bitsize" to 8, which means $2^8=256$ features will be created. The word/phrase in all the text will be hashed to 256 indices. The parameter "Hashing bitsize" ranges from 1 to 31. The word(s)/phrase(s) are less likely to be hashed into the same index if setting it to be a larger number.
- Third, set the parameter "N-grams" to 2. This value gets the occurrence frequency of unigrams (a feature for every single word) and bigrams (a feature for every pair of adjacent words) from the input text. The parameter "N-grams" ranges from 0 to 10, which indicates the maximum number of sequential words to be included in a feature.



The following figure shows what these new feature look like.



Conclusion

Engineered and selected features increase the efficiency of the training process, which attempts to extract the key information contained in the data. They also improve the power of these models to classify the input data accurately and to predict outcomes of interest more robustly.

Feature engineering and selection can also combine to make the learning more computationally tractable. It does so by enhancing and then reducing the number of features needed to calibrate or train a model.

Mathematically, the selected features are a minimal set of independent variables that explain the patterns in the data and predict outcomes successfully.

It's not always necessarily to perform feature engineering or feature selection. It depends on the data, the algorithm selected, and the objective of the experiment.

Next steps

To create features for data in specific environments, see the following articles:

- [Create features for data in SQL Server](#)
- [Create features for data in a Hadoop cluster using Hive queries](#)

Create features for data in SQL Server using SQL and Python

3/10/2022 • 5 minutes to read • [Edit Online](#)

This document shows how to generate features for data stored in a SQL Server VM on Azure that help algorithms learn more efficiently from the data. You can use SQL or a programming language like Python to accomplish this task. Both approaches are demonstrated here.

This task is a step in the [Team Data Science Process \(TDSP\)](#).

NOTE

For a practical example, you can consult the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Stored your data in SQL Server. If you have not, see [Move data to an Azure SQL Database for Azure Machine Learning](#) for instructions on how to move the data there.

Feature generation with SQL

In this section, we describe ways of generating features using SQL:

- [Count based Feature Generation](#)
- [Binning Feature Generation](#)
- [Rolling out the features from a single column](#)

NOTE

Once you generate additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, that can be joined with the original table.

Count based feature generation

This document demonstrates two ways of generating count features. The first method uses conditional sum and the second method uses the 'where` clause. These new features can then be joined with the original table (using primary key columns) to have count features alongside the original data.

```
select <column_name1>,<column_name2>,<column_name3>, COUNT(*) as Count_Features from <tablename> group by <column_name1>,<column_name2>,<column_name3>

select <column_name1>,<column_name2> , sum(1) as Count_Features from <tablename>
where <column_name3> = '<some_value>' group by <column_name1>,<column_name2>
```

Binning Feature Generation

The following example shows how to generate binned features by binning (using five bins) a numerical column

that can be used as a feature instead:

```
SELECT <column_name>, NTILE(5) OVER (ORDER BY <column_name>) AS BinNumber from <tablename>
```

Rolling out the features from a single column

In this section, we demonstrate how to roll out a single column in a table to generate additional features. The example assumes that there is a latitude or longitude column in the table from which you are trying to generate features.

Here is a brief primer on latitude/longitude location data (resourced from stackoverflow

<https://gis.stackexchange.com/questions/8650/how-to-measure-the-accuracy-of-latitude-and-longitude>). Here are some useful things to understand about location data before creating features from the field:

- The sign indicates whether we are north or south, east or west on the globe.
- A nonzero hundreds digit indicates longitude, not latitude is being used.
- The tens digit gives a position to about 1,000 kilometers. It gives useful information about what continent or ocean we are on.
- The units digit (one decimal degree) gives a position up to 111 kilometers (60 nautical miles, about 69 miles). It indicates, roughly, what large state or country/region we are in.
- The first decimal place is worth up to 11.1 km: it can distinguish the position of one large city from a neighboring large city.
- The second decimal place is worth up to 1.1 km: it can separate one village from the next.
- The third decimal place is worth up to 110 m: it can identify a large agricultural field or institutional campus.
- The fourth decimal place is worth up to 11 m: it can identify a parcel of land. It is comparable to the typical accuracy of an uncorrected GPS unit with no interference.
- The fifth decimal place is worth up to 1.1 m: it distinguishes trees from each other. Accuracy to this level with commercial GPS units can only be achieved with differential correction.
- The sixth decimal place is worth up to 0.11 m: you can use this level for laying out structures in detail, for designing landscapes, building roads. It should be more than good enough for tracking movements of glaciers and rivers. This goal can be achieved by taking painstaking measures with GPS, such as differentially corrected GPS.

The location information can be featurized by separating out region, location, and city information. Once can also call a REST endpoint, such as Bing Maps API (see <https://msdn.microsoft.com/library/ff701710.aspx> to get the region/district information).

```

select
    <location_columnname>
    ,round(<location_columnname>,0) as l1
    ,l2=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 1 then substring(PARSENAME(round(ABS(<location_columnname>) -
    FLOOR(ABS(<location_columnname>)),6),1),1,1) else '0' end
    ,l3=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 2 then substring(PARSENAME(round(ABS(<location_columnname>) -
    FLOOR(ABS(<location_columnname>)),6),1),2,1) else '0' end
    ,l4=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 3 then substring(PARSENAME(round(ABS(<location_columnname>) -
    FLOOR(ABS(<location_columnname>)),6),1),3,1) else '0' end
    ,l5=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 4 then substring(PARSENAME(round(ABS(<location_columnname>) -
    FLOOR(ABS(<location_columnname>)),6),1),4,1) else '0' end
    ,l6=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 5 then substring(PARSENAME(round(ABS(<location_columnname>) -
    FLOOR(ABS(<location_columnname>)),6),1),5,1) else '0' end
    ,l7=case when LEN (PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 6 then substring(PARSENAME(round(ABS(<location_columnname>),
    FLOOR(ABS(<location_columnname>)),6),1),6,1) else '0' end
from <tablename>

```

These location-based features can be further used to generate additional count features as described earlier.

TIP

You can programmatically insert the records using your language of choice. You may need to insert the data in chunks to improve write efficiency. [Here is an example of how to do this using pyodbc](#). Another alternative is to insert data in the database using [BCP utility](#)

Connecting to Azure Machine Learning

The newly generated feature can be added as a column to an existing table or stored in a new table and joined with the original table for machine learning. The resulting table may be then [saved as a dataset in Azure Machine Learning](#) and available for data science.

Using a programming language like Python

Using Python to generate features when the data is in SQL Server is similar to processing data in Azure blob using Python. For comparison, see [Process Azure Blob data in your data science environment](#). Load the data from the database into a pandas data frame to process it further. The process of connecting to the database and loading the data into the data frame is documented in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replace servername, dbname, username, and password with your specific values):

```

#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')

```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The following code reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame  
data_frame = pd.read_sql('''select <columnname1>, <columnname2>... from <tablename>''', conn)
```

Now you may work with the Pandas data frame as covered in topics [Create features for Azure blob storage data using Panda](#).

Create features for data in a Hadoop cluster using Hive queries

3/10/2022 • 7 minutes to read • [Edit Online](#)

This document shows how to create features for data stored in an Azure HDInsight Hadoop cluster using Hive queries. These Hive queries use embedded Hive User-Defined Functions (UDFs), the scripts for which are provided.

The operations needed to create features can be memory intensive. The performance of Hive queries becomes more critical in such cases and can be improved by tuning certain parameters. The tuning of these parameters is discussed in the final section.

Examples of the queries that are presented are specific to the [NYC Taxi Trip Data](#) scenarios are also provided in [GitHub repository](#). These queries already have data schema specified and are ready to be submitted to run. In the final section, parameters that users can tune so that the performance of Hive queries can be improved are also discussed.

This task is a step in the [Team Data Science Process \(TDSP\)](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Customize Azure HDInsight Hadoop Clusters for Advanced Analytics](#).
- The data has been uploaded to Hive tables in Azure HDInsight Hadoop clusters. If it has not, follow [Create and load data to Hive tables](#) to upload data to Hive tables first.
- Enabled remote access to the cluster. If you need instructions, see [Access the Head Node of Hadoop Cluster](#).

Feature generation

In this section, several examples of the ways in which features can be generating using Hive queries are described. Once you have generated additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, which can then be joined with the original table. Here are the examples presented:

1. [Frequency-based Feature Generation](#)
2. [Risks of Categorical Variables in Binary Classification](#)
3. [Extract features from Datetime Field](#)
4. [Extract features from Text Field](#)
5. [Calculate distance between GPS coordinates](#)

Frequency-based feature generation

It is often useful to calculate the frequencies of the levels of a categorical variable, or the frequencies of certain combinations of levels from multiple categorical variables. Users can use the following script to calculate these frequencies:

```

select
    a.<column_name1>, a.<column_name2>, a.sub_count/sum(a.sub_count) over () as frequency
from
(
    select
        <column_name1>,<column_name2>, count(*) as sub_count
    from <databasename>.<tablename> group by <column_name1>, <column_name2>
)a
order by frequency desc;

```

Risks of categorical variables in binary classification

In binary classification, non-numeric categorical variables must be converted into numeric features when the models being used only take numeric features. This conversion is done by replacing each non-numeric level with a numeric risk. This section shows some generic Hive queries that calculate the risk values (log odds) of a categorical variable.

```

set smooth_param1=1;
set smooth_param2=20;
select
    <column_name1>,<column_name2>,
    ln((sum_target+${hiveconf:smooth_param1})/(record_count-sum_target+${hiveconf:smooth_param2}-
${hiveconf:smooth_param1})) as risk
from
(
    select
        <column_name1>, <column_name2>, sum(binary_target) as sum_target, sum(1) as record_count
    from
    (
        select
            <column_name1>, <column_name2>, if(target_column>0,1,0) as binary_target
        from <databasename>.<tablename>
    )a
    group by <column_name1>, <column_name2>
)b

```

In this example, variables `smooth_param1` and `smooth_param2` are set to smooth the risk values calculated from the data. Risks have a range between -Inf and Inf. A risk > 0 indicates that the probability that the target is equal to 1 is greater than 0.5.

After the risk table is calculated, users can assign risk values to a table by joining it with the risk table. The Hive joining query was provided in previous section.

Extract features from datetime fields

Hive comes with a set of UDFs for processing datetime fields. In Hive, the default datetime format is 'yyyy-MM-dd 00:00:00' ('1970-01-01 12:21:32' for example). This section shows examples that extract the day of a month, the month from a datetime field, and other examples that convert a datetime string in a format other than the default format to a datetime string in default format.

```

select day(<datetime field>), month(<datetime field>)
from <databasename>.<tablename>;

```

This Hive query assumes that the `<datetime field>` is in the default datetime format.

If a datetime field is not in the default format, you need to convert the datetime field into Unix time stamp first, and then convert the Unix time stamp to a datetime string that is in the default format. When the datetime is in default format, users can apply the embedded datetime UDFs to extract features.

```
select from_unixtime(unix_timestamp(<datetime field>,'<pattern of the datetime field>'))
from <databasename>.<tablename>;
```

In this query, if the *<datetime field>* has the pattern like *03/26/2015 12:04:39*, the *<pattern of the datetime field>* 'should be '`'MM/dd/yyyy HH:mm:ss'`'. To test it, users can run

```
select from_unixtime(unix_timestamp('05/15/2015 09:32:10','MM/dd/yyyy HH:mm:ss'))
from hivesampletable limit 1;
```

The *hivesampletable* in this query comes preinstalled on all Azure HDInsight Hadoop clusters by default when the clusters are provisioned.

Extract features from text fields

When the Hive table has a text field that contains a string of words that are delimited by spaces, the following query extracts the length of the string, and the number of words in the string.

```
select length(<text field>) as str_len, size(split(<text field>,' ')) as word_num
from <databasename>.<tablename>;
```

Calculate distances between sets of GPS coordinates

The query given in this section can be directly applied to the NYC Taxi Trip Data. The purpose of this query is to show how to apply an embedded mathematical function in Hive to generate features.

The fields that are used in this query are the GPS coordinates of pickup and dropoff locations, named *pickup_longitude*, *pickup_latitude*, *dropoff_longitude*, and *dropoff_latitude*. The queries that calculate the direct distance between the pickup and dropoff coordinates are:

```
set R=3959;
set pi=radians(180);
select pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude,
    ${hiveconf:R}*2*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)
        *${hiveconf:pi}/180/2),2)-cos(pickup_latitude*${hiveconf:pi}/180)
        *cos(dropoff_latitude*${hiveconf:pi}/180)*pow(sin((dropoff_longitude-
        pickup_longitude)*${hiveconf:pi}/180/2),2)))
    /sqrt(pow(sin((dropoff_latitude-pickup_latitude)*${hiveconf:pi}/180/2),2)
    +cos(pickup_latitude*${hiveconf:pi}/180)*cos(dropoff_latitude*${hiveconf:pi}/180)*
    pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2))) as direct_distance
from nytaxi.trip
where pickup_longitude between -90 and 0
and pickup_latitude between 30 and 90
and dropoff_longitude between -90 and 0
and dropoff_latitude between 30 and 90
limit 10;
```

The mathematical equations that calculate the distance between two GPS coordinates can be found on the [Movable Type Scripts](#) site, authored by Peter Lapisu. In this Javascript, the function `toRad()` is just $\text{lat_or_lon}\pi/180$, which converts degrees to radians. Here, *lat_or_lon* is the latitude or longitude. Since Hive does not provide the function `atan2`, but provides the function `atan`, the `atan2` function is implemented by `atan` function in the above Hive query using the definition provided in [Wikipedia](#).

$$\text{atan2}(y, x) = 2 \arctan \frac{\sqrt{x^2 + y^2} - x}{y}.$$

A full list of Hive embedded UDFs can be found in the [Built-in Functions](#) section on the [Apache Hive wiki](#)).

Advanced topics: Tune Hive parameters to improve query speed

The default parameter settings of Hive cluster might not be suitable for the Hive queries and the data that the queries are processing. This section discusses some parameters that users can tune to improve the performance of Hive queries. Users need to add the parameter tuning queries before the queries of processing data.

1. **Java heap space:** For queries involving joining large datasets, or processing long records, **running out of heap space** is one of the common errors. This error can be avoided by setting parameters *mapreduce.map.java.opts* and *mapreduce.task.io.sort.mb* to desired values. Here is an example:

```
set mapreduce.map.java.opts=-Xmx4096m;
set mapreduce.task.io.sort.mb=-Xmx1024m;
```

This parameter allocates 4-GB memory to Java heap space and also makes sorting more efficient by allocating more memory for it. It is a good idea to play with these allocations if there are any job failure errors related to heap space.

2. **DFS block size:** This parameter sets the smallest unit of data that the file system stores. As an example, if the DFS block size is 128 MB, then any data of size less than and up to 128 MB is stored in a single block. Data that is larger than 128 MB is allotted extra blocks.
3. Choosing a small block size causes large overheads in Hadoop since the name node has to process many more requests to find the relevant block pertaining to the file. A recommended setting when dealing with gigabytes (or larger) data is:

```
set dfs.block.size=128m;
```

4. **Optimizing join operation in Hive:** While join operations in the map/reduce framework typically take place in the reduce phase, sometimes, enormous gains can be achieved by scheduling joins in the map phase (also called "mapjoins"). Set this option:

```
set hive.auto.convert.join=true;
```

5. **Specifying the number of mappers to Hive:** While Hadoop allows the user to set the number of reducers, the number of mappers is typically not be set by the user. A trick that allows some degree of control on this number is to choose the Hadoop variables *mapred.min.split.size* and *mapred.max.split.size* as the size of each map task is determined by:

```
num_maps = max(mapred.min.split.size, min(mapred.max.split.size, dfs.block.size))
```

Typically, the default value of:

- *mapred.min.split.size* is 0, that of
- *mapred.max.split.size* is **Long.MAX** and that of
- *dfs.block.size* is 64 MB.

As we can see, given the data size, tuning these parameters by "setting" them allows us to tune the number of mappers used.

6. Here are a few other more **advanced options** for optimizing Hive performance. These options allow you to set the memory allocated to map and reduce tasks, and can be useful in tweaking performance. Keep in mind that the *mapreduce.reduce.memory.mb* cannot be greater than the physical memory size of each worker node in the Hadoop cluster.

```
set mapreduce.map.memory.mb = 2048;
set mapreduce.reduce.memory.mb=6144;
set mapreduce.reduce.java.opts=-Xmx8192m;
set mapred.reduce.tasks=128;
set mapred.tasktracker.reduce.tasks.maximum=128;
```

Feature selection in the Team Data Science Process (TDSP)

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article explains the purposes of feature selection and provides examples of its role in the data enhancement process of machine learning. These examples are drawn from Azure Machine Learning Studio.

The engineering and selection of features is one part of the Team Data Science Process (TDSP) outlined in the article [What is the Team Data Science Process?](#). Feature engineering and selection are parts of the **Develop features** step of the TDSP.

- **feature engineering:** This process attempts to create additional relevant features from the existing raw features in the data, and to increase predictive power to the learning algorithm.
- **feature selection:** This process selects the key subset of original data features in an attempt to reduce the dimensionality of the training problem.

Normally **feature engineering** is applied first to generate additional features, and then the **feature selection** step is performed to eliminate irrelevant, redundant, or highly correlated features.

Filter features from your data - feature selection

Feature selection may be used for classification or regression tasks. The goal is to select a subset of the features from the original dataset that reduce its dimensions by using a minimal set of features to represent the maximum amount of variance in the data. This subset of features is used to train the model. Feature selection serves two main purposes.

- First, feature selection often increases classification accuracy by eliminating irrelevant, redundant, or highly correlated features.
- Second, it decreases the number of features, which makes the model training process more efficient.

Efficiency is important for learners that are expensive to train such as support vector machines.

Although feature selection does seek to reduce the number of features in the dataset used to train the model, it is not referred to by the term "dimensionality reduction". Feature selection methods extract a subset of original features in the data without changing them. Dimensionality reduction methods employ engineered features that can transform the original features and thus modify them. Examples of dimensionality reduction methods include principal component analysis (PCA), canonical correlation analysis, and singular value decomposition (SVD).

Among others, one widely applied category of feature selection methods in a supervised context is called "filter-based feature selection". By evaluating the correlation between each feature and the target attribute, these methods apply a statistical measure to assign a score to each feature. The features are then ranked by the score, which may be used to help set the threshold for keeping or eliminating a specific feature. Examples of statistical measures used in these methods include Pearson correlation coefficient (PCC), mutual information (MI), and the chi-squared test.

Azure Machine Learning Designer

One tool inside Azure Machine Learning is the [designer](#). Azure Machine Learning designer is a drag-and-drop interface used to train and deploy models in Azure Machine Learning. To manage features, there are [different tools available inside designer](#).

The [Filter Based Feature Selection component](#) in Azure Machine Learning designer helps you identify the columns in your input dataset that have the greatest predictive power.

The [Permutation Feature Importance component](#) in Azure Machine Learning designer computes a set of feature importance scores for your dataset; you then use these scores to help you determine the best features to use in a model.

Conclusion

Feature engineering and feature selection are two commonly engineering techniques to increase training efficiency. These techniques also improve the model's power to classify the input data accurately and to predict outcomes of interest more robustly. Feature engineering and selection can also combine to make the learning more computationally efficient by enhancing and then reducing the number of features needed to calibrate or train a model. Mathematically speaking, the features selected to train the model are a minimal set of independent variables that explain the maximum variance in the data to predict the outcome feature.

It is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data collected, the algorithm selected, and the objective of the experiment.

Deploy ML models to production to play an active role in making business decisions

3/10/2022 • 2 minutes to read • [Edit Online](#)

Production deployment enables a model to play an active role in a business. Predictions from a deployed model can be used for business decisions.

Production platforms

There are various approaches and platforms to put models into production. Here are a few options:

- [Where to deploy models with Azure Machine Learning](#)
- [Deployment of a model in SQL-server](#)

NOTE

Prior to deployment, one has to insure the latency of model scoring is low enough to use in production.

NOTE

For deployment from Azure Machine Learning, see [Deploy machine learning models to Azure](#).

A/B testing

When multiple models are in production, [A/B testing](#) may be used to compare model performance.

Next steps

- [What is the Team Data Science Process?](#)
- [Compare the machine learning products and technologies from Microsoft](#)
- [Machine learning at scale](#)

Choose an analytical data store in Azure

3/10/2022 • 5 minutes to read • [Edit Online](#)

In a [big data](#) architecture, there is often a need for an analytical data store that serves processed data in a structured format that can be queried using analytical tools. Analytical data stores that support querying of both hot-path and cold-path data are collectively referred to as the serving layer, or data serving storage.

The serving layer deals with processed data from both the hot path and cold path. In the [lambda architecture](#), the serving layer is subdivided into a *speed serving* layer, which stores data that has been processed incrementally, and a *batch serving* layer, which contains the batch-processed output. The serving layer requires strong support for random reads with low latency. Data storage for the speed layer should also support random writes, because batch loading data into this store would introduce undesired delays. On the other hand, data storage for the batch layer does not need to support random writes, but batch writes instead.

There is no single best data management choice for all data storage tasks. Different data management solutions are optimized for different tasks. Most real-world cloud apps and big data processes have a variety of data storage requirements and often use a combination of data storage solutions.

What are your options when choosing an analytical data store?

There are several options for data serving storage in Azure, depending on your needs:

- [Azure Synapse Analytics](#)
- [Azure Synapse Spark pools](#)
- [Azure Databricks](#)
- [Azure Data Explorer](#)
- [Azure SQL Database](#)
- [SQL Server in Azure VM](#)
- [HBase/Phoenix on HDInsight](#)
- [Hive LLAP on HDInsight](#)
- [Azure Analysis Services](#)
- [Azure Cosmos DB](#)

These options provide various database models that are optimized for different types of tasks:

- [Key/value](#) databases hold a single serialized object for each key value. They're good for storing large volumes of data where you want to get one item for a given key value and you don't have to query based on other properties of the item.
- [Document](#) databases are key/value databases in which the values are *documents*. A "document" in this context is a collection of named fields and values. The database typically stores the data in a format such as XML, YAML, JSON, or BSON, but may use plain text. Document databases can query on non-key fields and define secondary indexes to make querying more efficient. This makes a document database more suitable for applications that need to retrieve data based on criteria more complex than the value of the document key. For example, you could query on fields such as product ID, customer ID, or customer name.
- [Column-family](#) databases are key/value data stores that structure data storage into collections of related columns called column families. For example, a census database might have one group of columns for a person's name (first, middle, last), one group for the person's address, and one group for the person's profile information (date of birth, gender). The database can store each column family in a separate partition, while keeping all of the data for one person related to the same key. An application can read a single column family

without reading through all of the data for an entity.

- **Graph** databases store information as a collection of objects and relationships. A graph database can efficiently perform queries that traverse the network of objects and the relationships between them. For example, the objects might be employees in a human resources database, and you might want to facilitate queries such as "find all employees who directly or indirectly work for Scott."
- Telemetry and time series databases are an append-only collection of objects. Telemetry databases efficiently index data in a variety of column stores and in-memory structures, making them the optimal choice for storing and analyzing vast quantities of telemetry and time series data.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need serving storage that can serve as a hot path for your data? If yes, narrow your options to those that are optimized for a speed serving layer.
- Do you need massively parallel processing (MPP) support, where queries are automatically distributed across several processes or nodes? If yes, select an option that supports query scale out.
- Do you prefer to use a relational data store? If so, narrow your options to those with a relational database model. However, note that some non-relational stores support SQL syntax for querying, and tools such as PolyBase can be used to query non-relational data stores.
- Do you collect time series data? Do you use append-only data?

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CapabilitY	SQL DatabasE	Azure Synapse SQL Pool	Azure Synapse Spark Pool	Azure Data Explorer	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Is managed service	Yes	Yes	Yes	Yes	Yes ¹	Yes ¹	Yes	Yes
Primary database model	Relational (columnar format when using columnstore indexes)	Relational tables with columnar storage	Wide column store	Relational (column store), telemetry, and time series store	Wide column store	Hive/In-Memory	Tabular semantic models	Document store, graph, key-value store, wide column store
SQL language support	Yes	Yes	Yes	Yes	Yes (using Phoenix JDBC driver)	Yes	No	Yes

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Optimized for speed serving layer	Yes ²	Yes ³	Yes	Yes	Yes	Yes	No	Yes

[1] With manual configuration and scaling.

[2] Using memory-optimized tables and hash or nonclustered indexes.

[3] Supported as an Azure Stream Analytics output.

Scalability capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Redundant regional servers for high availability	Yes	No	No	Yes	Yes	No	No	Yes
Supports query scale out	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic scalability (scale up)	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Supports in-memory caching of data	Yes	Yes	Yes	Yes	No	Yes	Yes	No

Security capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Authentication	SQL / Azure Active Directory (Azure AD)	SQL / Azure AD	Azure AD	local / Azure AD ¹	local / Azure AD ¹	Azure AD	database users / Azure AD via access control (IAM)

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Data encryption at rest	Yes ²	Yes ²	Yes	Yes ¹	Yes ¹	Yes	Yes
Row-level security	Yes	Yes ³	No	Yes ¹	Yes ¹	Yes	No
Supports firewalls	Yes	Yes	Yes	Yes ⁴	Yes ⁴	Yes	Yes
Dynamic data masking	Yes	Yes	Yes	Yes ¹	Yes	No	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Requires using transparent data encryption (TDE) to encrypt and decrypt your data at rest.

[3] Filter predicates only. See [Row-Level Security](#)

[4] When used within an Azure Virtual Network. See [Extend Azure HDInsight using an Azure Virtual Network](#).

Choose a data analytics technology in Azure

3/10/2022 • 4 minutes to read • [Edit Online](#)

The goal of most big data solutions is to provide insights into the data through analysis and reporting. This can include preconfigured reports and visualizations, or interactive data exploration.

What are your options when choosing a data analytics technology?

There are several options for analysis, visualizations, and reporting in Azure, depending on your needs:

- [Power BI](#)
- [Jupyter Notebooks](#)
- [Zeppelin Notebooks](#)
- [Microsoft Azure Notebooks](#)

Power BI

[Power BI](#) is a suite of business analytics tools. It can connect to hundreds of data sources, and can be used for ad hoc analysis. See [this list](#) of the currently available data sources. Use [Power BI Embedded](#) to integrate Power BI within your own applications without requiring any additional licensing.

Organizations can use Power BI to produce reports and publish them to the organization. Everyone can create personalized dashboards, with governance and [security built in](#). Power BI uses [Azure Active Directory](#) (Azure AD) to authenticate users who log in to the Power BI service, and uses the Power BI login credentials whenever a user attempts to access resources that require authentication.

Jupyter Notebooks

[Jupyter Notebooks](#) provide a browser-based shell that lets data scientists create *notebook* files that contain Python, Scala, or R code and markdown text, making it an effective way to collaborate by sharing and documenting code and results in a single document.

Most varieties of HDInsight clusters, such as Spark or Hadoop, come [preconfigured with Jupyter notebooks](#) for interacting with data and submitting jobs for processing. Depending on the type of HDInsight cluster you are using, one or more kernels will be provided for interpreting and running your code. For example, Spark clusters on HDInsight provide Spark-related kernels that you can select from to execute Python or Scala code using the Spark engine.

Jupyter notebooks provide a great environment for analyzing, visualizing, and processing your data prior to building more advanced visualizations with a BI/reporting tool like Power BI.

Zeppelin Notebooks

[Zeppelin Notebooks](#) are another option for a browser-based shell, similar to Jupyter in functionality. Some HDInsight clusters come [preconfigured with Zeppelin notebooks](#). However, if you are using an [HDInsight Interactive Query](#) (Hive LLAP) cluster, [Zeppelin](#) is currently your only choice of notebook that you can use to run interactive Hive queries. Also, if you are using a [domain-joined HDInsight cluster](#), Zeppelin notebooks are the only type that enables you to assign different user logins to control access to notebooks and the underlying Hive tables.

Microsoft Azure Notebooks

[Azure Notebooks](#) is an online Jupyter Notebooks-based service that enables data scientists to create, run, and share Jupyter Notebooks in cloud-based libraries. Azure Notebooks provides execution environments for Python 2, Python 3, F#, and R, and provides several charting libraries for visualizing your data, such as ggplot,

matplotlib, bokeh, and seaborn.

Unlike Jupyter notebooks running on an HDInsight cluster, which are connected to the cluster's default storage account, Azure Notebooks does not provide any data. You must [load data](#) in a variety of ways, such downloading data from an online source, interacting with Azure Blobs or Table Storage, connecting to a SQL database, or loading data with the Copy Wizard for Azure Data Factory.

Key benefits:

- Free service—no Azure subscription required.
- No need to install Jupyter and the supporting R or Python distributions locally—just use a browser.
- Manage your own online libraries and access them from any device.
- Share your notebooks with collaborators.

Considerations:

- You will be unable to access your notebooks when offline.
- Limited processing capabilities of the free notebook service may not be enough to train large or complex models.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need to connect to numerous data sources, providing a centralized place to create reports for data spread throughout your domain? If so, choose an option that allows you to connect to 100s of data sources.
- Do you want to embed dynamic visualizations in an external website or application? If so, choose an option that provides embedding capabilities.
- Do you want to design your visualizations and reports while offline? If yes, choose an option with offline capabilities.
- Do you need heavy processing power to train large or complex AI models or work with very large data sets? If yes, choose an option that can connect to a big data cluster.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Connect to big data cluster for advanced processing	Yes	Yes	Yes	No
Managed service	Yes	Yes ¹	Yes ¹	Yes
Connect to 100s of data sources	Yes	No	No	No
Offline capabilities	Yes ²	No	No	No

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Embedding capabilities	Yes	No	No	No
Automatic data refresh	Yes	No	No	No
Access to numerous open source packages	No	Yes ³	Yes ³	Yes ⁴
Data transformation/cleaning options	Power Query, R	40 languages, including Python, R, Julia, and Scala	20+ interpreters, including Python, JDBC, and R	Python, F#, R
Pricing	Free for Power BI Desktop (authoring), see pricing for hosting options	Free	Free	Free
Multiuser collaboration	Yes	Yes (through sharing or with a multiuser server like JupyterHub)	Yes	Yes (through sharing)

[1] When used as part of a managed HDInsight cluster.

[2] With the use of Power BI Desktop.

[2] You can search the [Maven repository](#) for community-contributed packages.

[3] Python packages can be installed using either pip or conda. R packages can be installed from CRAN or GitHub. Packages in F# can be installed via nuget.org using the [Paket dependency manager](#).

Choose a batch processing technology in Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

Big data solutions often use long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files from scalable storage (like HDFS, Azure Data Lake Store, and Azure Storage), processing them, and writing the output to new files in scalable storage.

The key requirement of such batch processing engines is the ability to scale out computations, in order to handle a large volume of data. Unlike real-time processing, however, batch processing is expected to have latencies (the time between data ingestion and computing a result) that measure in minutes to hours.

Technology choices for batch processing

Azure Synapse Analytics

[Azure Synapse](#) is a distributed system designed to perform analytics on large data. It supports massive parallel processing (MPP), which makes it suitable for running high-performance analytics. Consider Azure Synapse when you have large amounts of data (more than 1 TB) and are running an analytics workload that will benefit from parallelism.

Azure Data Lake Analytics

[Data Lake Analytics](#) is an on-demand analytics job service. It is optimized for distributed processing of very large data sets stored in Azure Data Lake Store.

- Languages: [U-SQL](#) (including Python, R, and C# extensions).
- Integrates with Azure Data Lake Store, Azure Storage blobs, Azure SQL Database, and Azure Synapse.
- Pricing model is per-job.

HDInsight

HDInsight is a managed Hadoop service. Use it to deploy and manage Hadoop clusters in Azure. For batch processing, you can use [Spark](#), [Hive](#), [Hive LLAP](#), [MapReduce](#).

- Languages: R, Python, Java, Scala, SQL
- Kerberos authentication with Active Directory, Apache Ranger based access control
- Gives you full control of the Hadoop cluster

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform. You can think of it as "Spark as a service." It's the easiest way to use Spark on the Azure platform.

- Languages: R, Python, Java, Scala, Spark SQL
- Fast cluster start times, autotermination, autoscaling.
- Manages the Spark cluster for you.
- Built-in integration with Azure Blob Storage, Azure Data Lake Storage (ADLS), Azure Synapse, and other services. See [Data Sources](#).
- User authentication with Azure Active Directory.
- Web-based [notebooks](#) for collaboration and data exploration.
- Supports [GPU-enabled clusters](#)

Azure Distributed Data Engineering Toolkit

The [Distributed Data Engineering Toolkit](#) (AZTK) is a tool for provisioning on-demand Spark on Docker clusters

in Azure.

AZTK is not an Azure service. Rather, it's a client-side tool with a CLI and Python SDK interface, that's built on Azure Batch. This option gives you the most control over the infrastructure when deploying a Spark cluster.

- Bring your own Docker image.
- Use low-priority VMs for an 80% discount.
- Mixed mode clusters that use both low-priority and dedicated VMs.
- Built in support for Azure Blob Storage and Azure Data Lake connection.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Do you want to author batch processing logic declaratively or imperatively?
- Will you perform batch processing in bursts? If yes, consider options that let you auto-terminate the cluster or whose pricing model is per batch job.
- Do you need to query relational data stores along with your batch processing, for example to look up reference data? If yes, consider the options that enable querying of external relational stores.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT	AZURE DATABRICKS
Is managed service	Yes	Yes	Yes ¹	Yes
Relational data store	Yes	Yes	No	No
Pricing model	Per batch job	By cluster hour	By cluster hour	Databricks Unit ² + cluster hour

[1] With manual configuration.

[2] A Databricks Unit (DBU) is a unit of processing capability per hour.

Capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT WITH SPARK	HDINSIGHT WITH HIVE	HDINSIGHT WITH HIVE LLAP	AZURE DATABRICKS
Autoscaling	No	No	Yes	Yes	Yes	Yes
Scale-out granularity	Per job	Per cluster	Per cluster	Per cluster	Per cluster	Per cluster
In-memory caching of data	No	Yes	Yes	No	Yes	Yes

Capability	Azure Data Lake Analytics	Azure Synapse	HDIInsight with Spark	HDIInsight with Hive	HDIInsight with Hive LLAP	Azure Databricks
Query from external relational stores	Yes	No	Yes	No	No	Yes
Authentication	Azure AD	SQL / Azure AD	No	Azure AD ¹	Azure AD ¹	Azure AD
Auditing	Yes	Yes	No	Yes ¹	Yes ¹	Yes
Row-level security	No	Yes ²	No	Yes ¹	Yes ¹	No
Supports firewalls	Yes	Yes	Yes	Yes ³	Yes ³	No
Dynamic data masking	No	Yes	No	Yes ¹	Yes ¹	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Filter predicates only. See [Row-Level Security](#)

[3] Supported when [used within an Azure Virtual Network](#).

Next steps

- [Analytics architecture design](#)
- [Choose an analytical data store in Azure](#)
- [Choose a data analytics technology in Azure](#)
- [Analytics end-to-end with Azure Synapse](#)

High-performance computing (HPC) on Azure

3/10/2022 • 7 minutes to read • [Edit Online](#)

Introduction to HPC

High-performance computing (HPC), also called "big compute", uses a large number of CPU or GPU-based computers to solve complex mathematical tasks.

Many industries use HPC to solve some of their most difficult problems. These include workloads such as:

- Genomics
- Oil and gas simulations
- Finance
- Semiconductor design
- Engineering
- Weather modeling

How is HPC different on the cloud?

One of the primary differences between an on-premises HPC system and one in the cloud is the ability for resources to dynamically be added and removed as they're needed. Dynamic scaling removes compute capacity as a bottleneck and instead allow customers to right size their infrastructure for the requirements of their jobs.

The following articles provide more detail about this dynamic scaling capability.

- [Big Compute Architecture Style](#)
- [Autoscaling best practices](#)

Implementation checklist

As you're looking to implement your own HPC solution on Azure, ensure you've reviewed the following topics:

- Choose the appropriate [architecture](#) based on your requirements
- Know which [compute](#) options is right for your workload
- Identify the right [storage](#) solution that meets your needs
- Decide how you're going to [manage](#) all your resources
- Optimize your [application](#) for the cloud
- [Secure](#) your Infrastructure

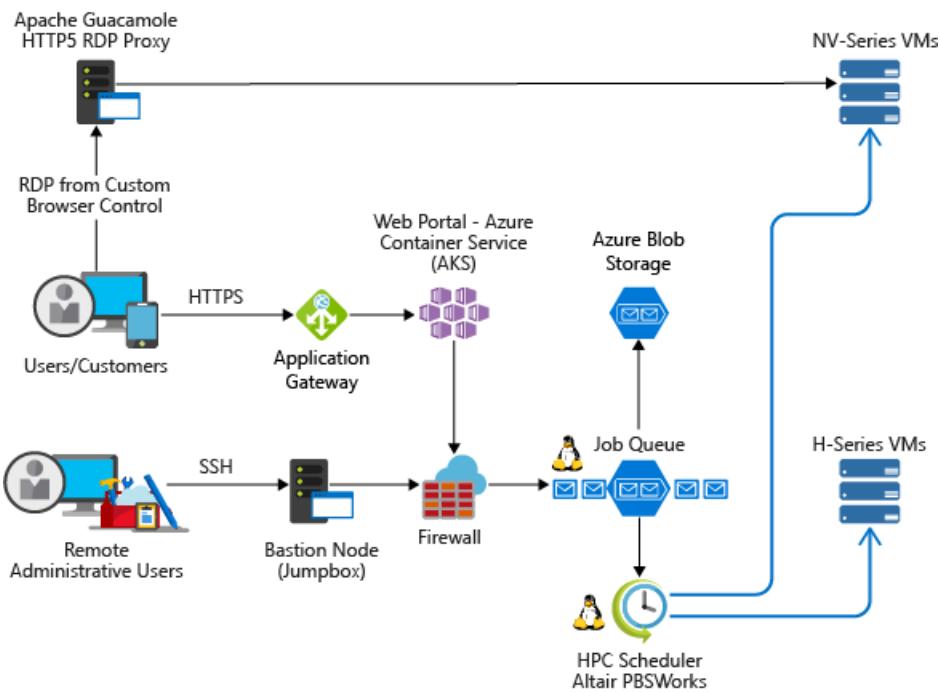
Infrastructure

There are many infrastructure components that are necessary to build an HPC system. Compute, storage, and networking provide the underlying components, no matter how you choose to manage your HPC workloads.

Example HPC architectures

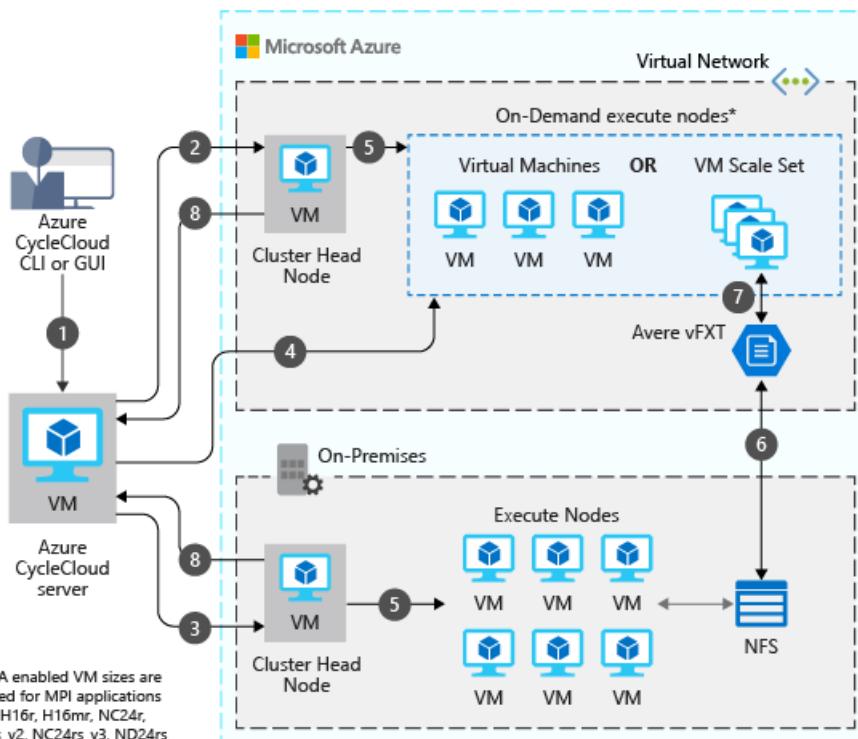
There are many different ways to design and implement your HPC architecture on Azure. HPC applications can scale to thousands of compute cores, extend on-premises clusters, or run as a 100% cloud-native solution.

The following scenarios outline a few of the common ways HPC solutions are built.



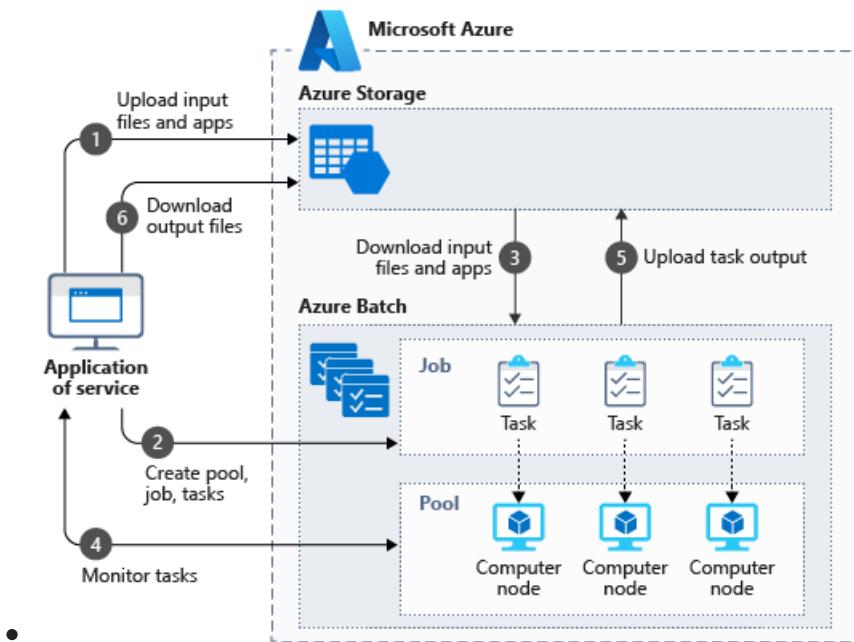
Computer-aided engineering services on Azure

Provide a software-as-a-service (SaaS) platform for computer-aided engineering (CAE) on Azure.



Computational fluid dynamics (CFD) simulations on Azure

Execute computational fluid dynamics (CFD) simulations on Azure.



3D video rendering on Azure

Run native HPC workloads in Azure using the Azure Batch service

Compute

Azure offers a range of sizes that are optimized for both CPU & GPU intensive workloads.

CPU-based virtual machines

- [Linux VMs](#)
- [Windows VMs](#)

GPU-enabled virtual machines

N-series VMs feature NVIDIA GPUs designed for compute-intensive or graphics-intensive applications including artificial intelligence (AI) learning and visualization.

- [Linux VMs](#)
- [Windows VMs](#)

Storage

Large-scale Batch and HPC workloads have demands for data storage and access that exceed the capabilities of traditional cloud file systems. There are many solutions that manage both the speed and capacity needs of HPC applications on Azure:

- [Avere vFXT](#) for faster, more accessible data storage for high-performance computing at the edge
- [Azure NetApp Files](#)
- [BeeGFS](#)
- [Storage Optimized Virtual Machines](#)
- [Blob, table, and queue storage](#)
- [Azure SMB File storage](#)

For more information comparing Lustre, GlusterFS, and BeeGFS on Azure, review the [Parallel File Systems on Azure](#) e-book and the [Lustre on Azure](#) blog.

Networking

H16r, H16mr, A8, and A9 VMs can connect to a high throughput back-end RDMA network. This network can improve the performance of tightly coupled parallel applications running under Microsoft MPI or Intel MPI.

- [RDMA Capable Instances](#)

- [Virtual Network](#)
- [ExpressRoute](#)

Management

Do-it-yourself

Building an HPC system from scratch on Azure offers a significant amount of flexibility, but it is often very maintenance intensive.

1. Set up your own cluster environment in Azure virtual machines or [virtual machine scale sets](#).
2. Use Azure Resource Manager templates to deploy leading [workload managers](#), infrastructure, and [applications](#).
3. Choose HPC and GPU [VM sizes](#) that include specialized hardware and network connections for MPI or GPU workloads.
4. Add [high-performance storage](#) for I/O-intensive workloads.

Hybrid and cloud Bursting

If you have an existing on-premises HPC system that you'd like to connect to Azure, there are several resources to help get you started.

First, review the [Options for connecting an on-premises network to Azure](#) article in the documentation. From there, you may want information on these connectivity options:

- 

[Connect an on-premises network to Azure using a VPN gateway](#)

This reference architecture shows how to extend an on-premises network to Azure, using a site-to-site virtual private network (VPN).

- 

[Connect an on-premises network to Azure using ExpressRoute](#)

ExpressRoute connections use a private, dedicated connection through a third-party connectivity provider. The private connection extends your on-premises network into Azure.

- 

[Connect an on-premises network to Azure using ExpressRoute with VPN failover](#)

Implement a highly available and secure site-to-site network architecture that spans an Azure virtual network and an on-premises network connected using ExpressRoute with VPN gateway failover.

Once network connectivity is securely established, you can start using cloud compute resources on-demand with the bursting capabilities of your existing [workload manager](#).

Marketplace solutions

There are many workload managers offered in the [Azure Marketplace](#).

- [RogueWave CentOS-based HPC](#)
- [SUSE Linux Enterprise Server for HPC](#)
- [TIBCO Datasyncapse GridServer](#)
- [Azure Data Science VM for Windows and Linux](#)

- [D3View](#)
- [UberCloud](#)

Azure Batch

[Azure Batch](#) is a platform service for running large-scale parallel and high-performance computing (HPC) applications efficiently in the cloud. Azure Batch schedules compute-intensive work to run on a managed pool of virtual machines, and can automatically scale compute resources to meet the needs of your jobs.

SaaS providers or developers can use the Batch SDKs and tools to integrate HPC applications or container workloads with Azure, stage data to Azure, and build job execution pipelines.

Azure CycleCloud

[Azure CycleCloud](#) Provides the simplest way to manage HPC workloads using any scheduler (like Slurm, Grid Engine, HPC Pack, HTCondor, LSF, PBS Pro, or Symphony), on Azure

CycleCloud allows you to:

- Deploy full clusters and other resources, including scheduler, compute VMs, storage, networking, and cache
- Orchestrate job, data, and cloud workflows
- Give admins full control over which users can run jobs, as well as where and at what cost
- Customize and optimize clusters through advanced policy and governance features, including cost controls, Active Directory integration, monitoring, and reporting
- Use your current job scheduler and applications without modification
- Take advantage of built-in autoscaling and battle-tested reference architectures for a wide range of HPC workloads and industries

Workload managers

The following are examples of cluster and workload managers that can run in Azure infrastructure. Create stand-alone clusters in Azure VMs or burst to Azure VMs from an on-premises cluster.

- Alces Flight Compute
- [TIBCO DataSynapse GridServer](#)
- [Bright Cluster Manager](#)
- [IBM Spectrum Symphony and Symphony LSF](#)
- [Altair PBS Works](#)
- Rescale
- [Univa Grid Engine](#)
- [Microsoft HPC Pack](#)
 - [HPC Pack for Windows](#)
 - [HPC Pack for Linux](#)

Containers

Containers can also be used to manage some HPC workloads. Services like the Azure Kubernetes Service (AKS) makes it simple to deploy a managed Kubernetes cluster in Azure.

- [Azure Kubernetes Service \(AKS\)](#)
- [Container Registry](#)

Cost management

Managing your HPC cost on Azure can be done through a few different ways. Ensure you've reviewed the [Azure purchasing options](#) to find the method that works best for your organization.

Security

For an overview of security best practices on Azure, review the [Azure Security Documentation](#).

In addition to the network configurations available in the [Cloud Bursting](#) section, you may want to implement a hub/spoke configuration to isolate your compute resources:

- 

Implement a hub-spoke network topology in Azure

The hub is a virtual network (VNet) in Azure that acts as a central point of connectivity to your on-premises network. The spokes are VNets that peer with the hub, and can be used to isolate workloads.

- 

Implement a hub-spoke network topology with shared services in Azure

This reference architecture builds on the hub-spoke reference architecture to include shared services in the hub that can be consumed by all spokes.

HPC applications

Run custom or commercial HPC applications in Azure. Several examples in this section are benchmarked to scale efficiently with additional VMs or compute cores. Visit the [Azure Marketplace](#) for ready-to-deploy solutions.

NOTE

Check with the vendor of any commercial application for licensing or other restrictions for running in the cloud. Not all vendors offer pay-as-you-go licensing. You might need a licensing server in the cloud for your solution, or connect to an on-premises license server.

Engineering applications

- [Altair RADIOSS](#)
- [ANSYS CFD](#)
- [MATLAB Distributed Computing Server](#)
- [StarCCM+](#)

Graphics and rendering

- [Autodesk Maya, 3ds Max, and Arnold](#) on Azure Batch

AI and deep learning

- [Microsoft Cognitive Toolkit](#)
- [Batch Shipyard recipes for deep learning](#)

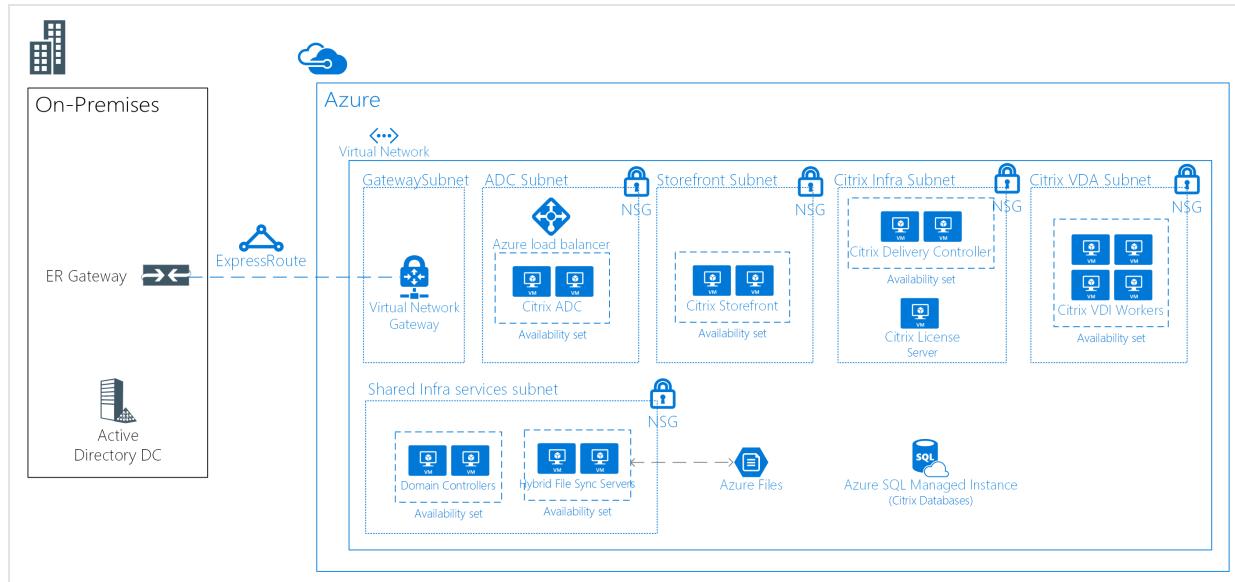
MPI providers

- [Microsoft MPI](#)

Remote visualization

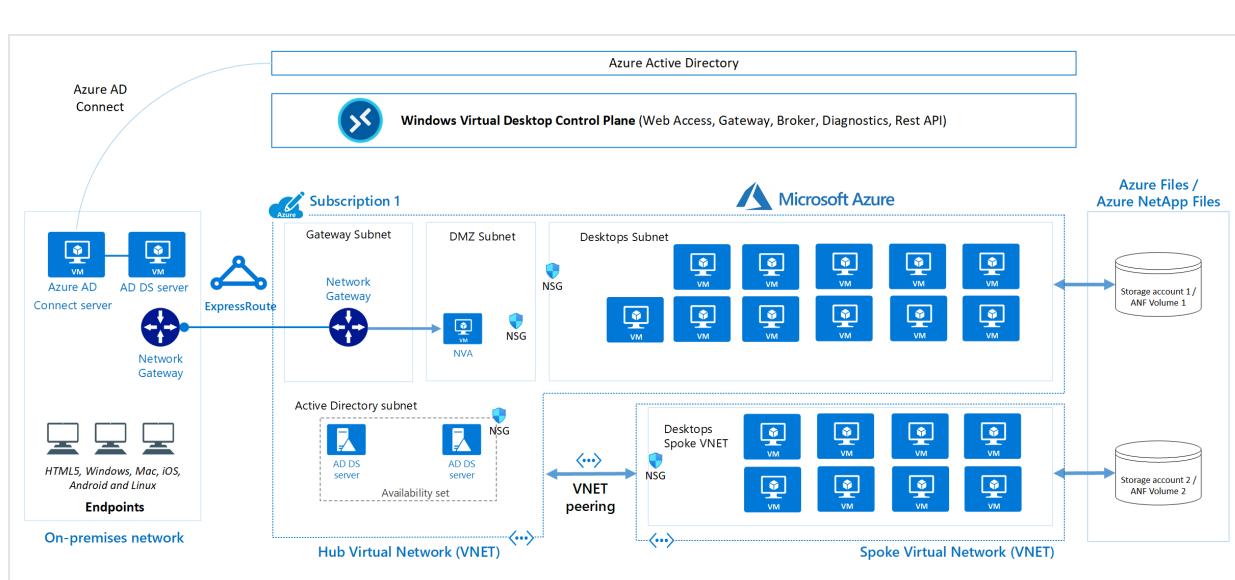
Run GPU-powered virtual machines in Azure in the same region as the HPC output for the lowest latency, access, and to visualize remotely through Azure Virtual Desktop, Citrix, or VMware Horizon.

- GPU-optimized virtual machine sizes
- Configure GPU acceleration for Azure Virtual Desktop



Linux virtual desktops with Citrix

Build a VDI environment for Linux desktops using Citrix on Azure.



Windows desktops using Azure Virtual Desktop on Azure

Build a VDI environment for Windows desktops using Azure Virtual Desktop on Azure.

Performance benchmarks

- Compute benchmarks

Customer stories

There are many customers who have seen great success by using Azure for their HPC workloads. You can find a few of these customer case studies below:

- AXA Global P&C
- Axioma

- [d3View](#)
- [EFS](#)
- [Hymans Robertson](#)
- [MetLife](#)
- [Microsoft Research](#)
- [Milliman](#)
- [Mitsubishi UFJ Securities International](#)
- [NeuroInitiative](#)
- [Schlumberger](#)
- [Towers Watson](#)

Other important information

- Ensure your [vCPU quota](#) has been increased before attempting to run large-scale workloads.

Next steps

For the latest announcements, see:

- [Microsoft HPC and Batch team blog](#)
- Visit the [Azure blog](#).

Microsoft Batch Examples

These tutorials will provide you with details on running applications on Microsoft Batch

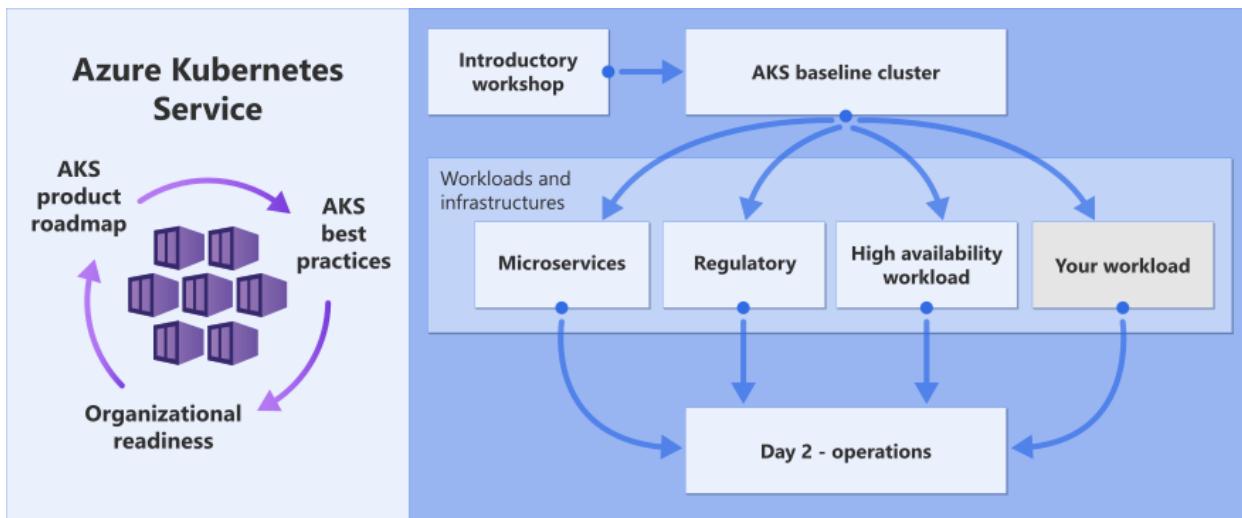
- [Get started developing with Batch](#)
- [Use Azure Batch code samples](#)
- [Use low-priority VMs with Batch](#)
- [Run containerized HPC workloads with Batch Shipyard](#)
- [Run parallel R workloads on Batch](#)
- [Run on-demand Spark jobs on Batch](#)
- [Use compute-intensive VMs in Batch pools](#)

Azure Kubernetes Service (AKS) architecture design

3/10/2022 • 4 minutes to read • [Edit Online](#)

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. [Azure Kubernetes Service \(AKS\)](#) makes it simple to deploy a managed Kubernetes cluster in Azure.

Organizations are at various points in their understanding, rationalizing, and adoption of Kubernetes on Azure. Your organization's journey will likely follow a similar path to many other technologies you've adopted; learning, aligning your organization around roles & responsibilities, and deploying production-ready workloads. From there, you'll iterate; growing your product as your customer and business demands change.



Introduction to Azure Kubernetes Service (AKS)

If you're new to Kubernetes or AKS, the best place to learn about the service is with Microsoft Learn. Microsoft Learn is a free, online training platform that provides interactive learning for Microsoft products and more. The [Introduction to Kubernetes on Azure](#) learning path will provide you with foundational knowledge that will take you through core concepts of containers, AKS cluster management, and workload deployment.

[Introduction to Kubernetes on Azure](#)

Path to production

You understand the benefits and trade-offs of Kubernetes, and have decided that AKS is the best Azure compute platform for your workload. Your organizational controls have been put into place; you're ready to learn how to deploy production-ready clusters for your workload.

Microsoft's AKS Baseline Cluster is the starting point to help you build production-ready AKS clusters.

[Microsoft's AKS Baseline Cluster](#)

We recommend you start from the baseline implementation and modify it to align to your workload's specific needs.

Suite of baseline implementations

We've provided a set of more baseline implementations to illustrate how you can adopt and configure components of AKS Baseline Cluster for various scenarios.

Microservices

When running microservices in the baseline cluster, you'll need to configure network policies, pod autoscaling, and set up distributed tracing for observability.

[Microservices architecture using the baseline implementation](#)

High security compliance

If you need a regulated environment, make the baseline implementation highly secure and restrict interactions to and from of the cluster. This use case is demonstrated in a cluster that's designed to run a PCI-DSS 3.2.1 workload.

[Regulated baseline cluster for PCI-DSS 3.2.1](#)

Business continuity and disaster recovery

A resilient solution needs multiple instances of the baseline cluster across regions in an active/active and highly available configuration.

[Baseline for multiregion clusters](#)

Best practices

As organizations such as yours have adopted Azure, the [Cloud Adoption Framework](#) provides them prescriptive guidance as they move between the phases of the cloud adoption lifecycle. The Cloud Adoption Framework includes tools, programs, and content to simplify adoption of Kubernetes and related cloud-native practices at scale.

[Kubernetes in the Cloud Adoption Framework](#)

As part of on going operations, you may wish to spot check your cluster against current recommended best practices. The best place to start is to ensure your cluster is aligned with Microsoft's [AKS Baseline Cluster](#).

See [Best Practices for Cluster Operations](#) and [Best Practices for AKS Workloads](#).

You may also consider evaluating a community-driven utility like [The AKS Checklist](#) as a way of organizing and tracking your alignment to these best practices.

Operations guide

Getting your workload deployed on AKS is a great milestone and this is when [day-2 operations](#) are going to be top-of-mind. [Microsoft's AKS Day 2 Operations Guide](#) was built for your ease of reference. This will help ensure you are ready to meet the demands of your customers and ensure you are prepared for break-fix situations via optimized triage processes.

[Microsoft's AKS Day 2 Operations Guide](#)

Stay current with AKS

Kubernetes and AKS are both moving fast. The platform is evolving and just knowing what's on the roadmap might help you make architectural decisions and understand planned deprecations; consider bookmarking it.

[AKS product roadmap](#)

Additional resources

The typical AKS solution journey shown ranges from learning about AKS to growing your existing clusters to meet new product and customer demands. However, you might also just be looking for additional reference and

supporting material to help along the way for your specific situation.

Example solutions

If you're seeking additional references that use AKS as their foundation, here are a few to consider.

- [Microservices architecture on AKS](#)
- [Secure DevOps for AKS](#)
- [Building a telehealth system](#)
- [CI/CD pipeline for container-based workloads](#)

Azure Arc-enabled Kubernetes

Azure Kubernetes Service offers you a managed Kubernetes experience on Azure, however there are workloads or situations that might be best suited for placing your own Kubernetes clusters under [Azure Arc-enabled Kubernetes](#) management. This includes your clusters such as RedHat OpenShift, RedHat RKE, and Canonical Charmed Kubernetes. Azure Arc management can also be used with [Cluster API provider Azure](#) clusters to benefit from the Azure Resource Manager representation of the cluster and availability of cluster extensions like Azure Monitor container insights and Azure Policy. Azure Arc-enabled Kubernetes can also be used with [AKS on Azure Stack HCI clusters](#) and with Kubernetes clusters running on other cloud providers.

[Azure Arc-enabled Kubernetes](#)

Managed service provider

If you're a managed service provider, you already use Azure Lighthouse to manage resources for multiple customers. Azure Kubernetes Service supports Azure Lighthouse so that you can manage hosted Kubernetes environments and deploy containerized applications within your customers' tenants.

[AKS with Azure Lighthouse](#)

AWS or Google Cloud professionals

These articles provide service mapping and comparison between Azure and other cloud services. This reference can help you ramp up quickly on Azure.

- [Containers and container orchestrators for AWS Professionals](#)
- [Containers and container orchestrators for Google Cloud Professionals](#)

Azure Kubernetes Services (AKS) day-2 operations guide

3/10/2022 • 2 minutes to read • [Edit Online](#)

After you release an Azure Kubernetes Service (AKS)-hosted application, prepare for *day-2 operations*. Day-2 operations include triage, ongoing maintenance of deployed assets, rolling out upgrades, and troubleshooting.

Day-2 operations help you:

- Keep up to date with your service-level agreement (SLA) or service-level objective (SLO) requirements.
- Troubleshoot customer support requests.
- Stay current with the latest platform features and security updates.
- Plan for future growth.

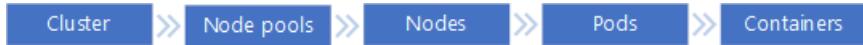
Prerequisites

The Day-2 operations guide assumes that you've deployed the [Azure Kubernetes Service \(AKS\) baseline architecture](#) as an example of a production cluster.

Triage practices

3/10/2022 • 2 minutes to read • [Edit Online](#)

It's often challenging to do root-cause analysis given the different aspects of an AKS cluster. When triaging issues, consider a top-down approach on the cluster hierarchy. Start at the cluster level and drill down if necessary.



In the triage practices series, we'll walk you through the thought process of this approach. The articles show examples using a set of tools and dashboards, and how they can highlight some symptoms.

Common causes addressed in this series include:

- Network and connectivity problems caused by improper configuration.
- Control plane to node communication is broken.
- Kubelet pressures caused by insufficient compute, memory, or storage resources.
- DNS resolution issues.
- Node health is running out of disk IOPS.
- Admission control pipeline is blocking a large number of requests to the API server.
- The cluster doesn't have permissions to pull from the appropriate container registry.

This series isn't intended to resolve specific issues. For information about troubleshooting specific issues, see [AKS Common Issues](#).

In the triage practices series

STEP	DESCRIPTION
1- Check the AKS cluster health	Start by checking the cluster the health of the overall cluster and networking.
2- Examine the node and pod health	Check the health of the AKS worker nodes.
3- Check the workload deployments	Check to see that all deployments and daemonSets are running.
4- Validate the admission controllers	Check whether the admission controllers are working as expected.
5- Verify the connection to the container registry	Verify the connection to the container registry.

Related links

- [Day-2 operations](#)
- [AKS periscope](#)
- [AKS roadmap](#)
- [AKS documentation](#)

Check the AKS cluster health

3/10/2022 • 2 minutes to read • [Edit Online](#)

Start by checking the health of the overall cluster and networking.

This article is part of a series. Read the introduction [here](#).

Tools:

AKS Diagnostics. In Azure portal, navigate to the AKS cluster resource. Select **Diagnose and solve problems**.

Dashboard > Kubernetes services > akskh20200929

The screenshot shows the 'akskh20200929 | Diagnose and solve problems' blade in the Azure portal. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems (which is selected and highlighted in grey), and Security. Under 'Kubernetes resources', there are links for Namespaces (preview), Workloads (preview), Services and ingresses (preview), Storage (preview), Configuration (preview), and Settings. The main content area has a search bar at the top right. Below it, a section titled 'Azure Kubernetes Service Diagnostics (Preview)' explains its purpose: 'Use Azure Kubernetes Service Diagnostics (Preview) to investigate how your cluster is performing, diagnose issues, and discover how to improve its reliability. Select the problem category that best matches the information or tool that you're interested in:'. Two cards are visible: 'Cluster Insights' (blue background) and 'Networking' (orange background). Both cards contain descriptive text and a list of keywords.

Category	Description	Keywords
Cluster Insights	Is your cluster experiencing failures or unresponsiveness? Investigate and discover issues that may cause your cluster to no longer be manageable.	Failed State, Node Readiness, Node Health, Scaling, CRUD, Identity, Certificates
Networking	Are you having networking issues with your cluster? Check out your cluster's network configuration and discover issues that may affect your cluster's traffic.	Network Configuration, Subnet, DNS, FQDN, VNet

Diagnostics shows a list of results from various test runs. If there are any issues found, **More info** can show you information about the underlying issue.

This image indicates that network and connectivity issues are caused by Azure CNI subnet configuration.

The screenshot shows the 'Issues' section of the AKS Diagnose and solve problems blade. It lists one issue under 'Network and Connectivity': 'We found Azure CNI subnet configuration that may cause issues with certain cluster operations' with a 'More Info' link. Below this, there is a section for 'Successful Checks (17)' with a green checkmark icon. It lists several categories: Application and Development, Monitoring and Logging, Cluster Management, CoreDNS Upgrade Breaking Change, and Kubelet Update Bug, each with a 'More Info' link.

Category	Description	Link
Network and Connectivity	We found Azure CNI subnet configuration that may cause issues with certain cluster operations	More Info
Successful Checks (17)	Tests that succeeded	
Application and Development	Our analysis did not find any issues in this category. Please click for recommended next steps.	More Info
Monitoring and Logging	Our analysis did not find any issues in this category. Please click for recommended next steps.	More Info
Cluster Management	Our analysis did not find any issues in this category. Please click for recommended next steps.	More Info
CoreDNS Upgrade Breaking Change	CoreDNS is not using the proxy plugin	More Info
Kubelet Update Bug	This Cluster Is Not Affected By This Bug	More Info

- > ⓘ Cluster uses User Defined Routing
- > ⓘ Allocated Outbound Ports is set to the default value
- > ⚠ We found Azure CNI subnet configuration that may cause issues with certain cluster operations
 - ✓ No 'SubnetIsFull' errors were found for the given time period from 2020-10-18 22:25:00 to 2020-10-19 22:25:00.
- > ✓ DNS resolved with no issues

To learn more about this feature, see [Azure Kubernetes Service Diagnostics overview](#).

Next steps

[Examine the node and pod health](#)

Examine the node and pod health

3/10/2022 • 5 minutes to read • [Edit Online](#)

If the cluster checks are clear, check the health of the AKS worker nodes. Determine the reason for the unhealthy node and resolve the issue.

This article is part of a series. Read the introduction [here](#).

1- Check the overall health of the worker nodes

A node can be unhealthy because of various reasons. A common reason is when the control plane to node communication is broken as a result of misconfiguration of routing and firewall rules. As a fix, you can allow the necessary ports and fully qualified domain names through the firewall according to the [AKS egress traffic guidance](#). Another reason can be kubelet pressures. In that case, add more compute, memory, or storage resources.

Tools:

You can check node health in one of these ways:

- **Azure Monitor - Containers health view.** In Azure portal, open Azure Monitor. Select **Containers**.

On the right pane, select **Monitored clusters**. Select the cluster to view the health of the nodes, user pods, and system pods.

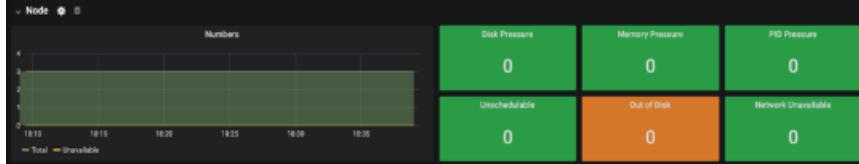
The screenshot shows the Azure Monitor interface for containers. On the left, there's a sidebar with links like Overview, Activity log, Alerts, Metrics, Logs, Service Health, Workbooks, and Insights. Under Insights, 'Containers' is selected. The main area has tabs for 'Getting started', 'Monitored clusters (1)', and 'Unmonitored clusters (1)'. Below this is a 'Cluster Status Summary' section with counts for Total (2), Critical (0), Warning (0), Unknown (0), Healthy (1), and Unmonitored (1). A table below lists the cluster details: Cluster Name (akskh20200929), Cluster Type (AKS), Version (1.18.8), Status (Healthy), Nodes (4 / 4), User Pods (41 / 41), and System Pods (31 / 31).

- **AKS - Nodes view:** In Azure portal, open navigate to the cluster. Select **Insights** under **Monitoring**. View **Nodes** on the right pane.

The screenshot shows the Azure Monitor Container Insights interface. On the left, there's a sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Kubernetes resources (Namespaces, Workloads, Services and ingresses, Storage, Configuration), Settings (Node pools, Configuration, Scale, Networking, Dev Spaces, Deployment center (preview), Policies, Properties, Locks), and Monitoring (Insights). The main area is titled 'Nodes' and shows a table of nodes. The table has columns: NAME, STATUS, 95TH %, CONTAINERS, UPTIME, CONTROLLER, and TREND 95TH % (1 BAR = 15M). There are four items listed:

NAME	STATUS	95TH %	CONTAINERS	UPTIME	CONTROLLER	TREND 95TH % (1 BAR = 15M)
aks-nodepool1-2885...	Ok	27%	534 mc	14	1 hour	-
aks-npuser1-288570...	Ok	13%	504 mc	39	1 hour	-
aks-nodepool1-2885...	Ok	12%	234 mc	22	1 hour	-
aks-npuser1-288570...	Ok	9%	374 mc	40	1 hour	-

- Prometheus and Grafana Dashboard. Open the Node Conditions dashboard.



2- Verify the control plane and worker node connectivity

If worker nodes are healthy, examine the connectivity between the managed AKS control plane and the worker nodes. Depending on the age and type of cluster configuration, the connectivity pods are either **tunneelfront** or **aks-link**, and located in the **kube-system** namespace.

Tools:

- `kubectl`
- Azure Monitor container insights

```
(⎈ | akskh20200929-admin:kube-system): repos → kubectl get pods | grep aks-link
aks-link-9947495dd-cqp2v
2/2      Running   0          26m
```

If **tunneelfront** or **aks-link** connectivity is not working, establish connectivity after checking that the appropriate AKS egress traffic rules have been allowed. Here are the steps:

1. Restart **tunneelfront** or **aks-link**.

```
kubectl rollout restart deploy aks-link
```

If restarting the pods doesn't fix the connection, continue to the next step.

2. Check the logs and look for abnormalities. This output shows logs for a working connection.

```
kubectl logs -l app=aks-link -c openvpn-client --tail=50
```

```
(⎈ | akskh20200929-admin:kube-system): repos → kubectl logs aks-link-9947495dd-cqp2v openvpn-client
Tue Sep 29 13:16:29 2020 WARNING: file '/etc/openvpn/certs/client.key' is group or others accessible
Tue Sep 29 13:16:29 2020 OpenVPN 2.4.4 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on May 14 2019
Tue Sep 29 13:16:29 2020 library versions: OpenSSL 1.1.1 11 Sep 2018, LZO 2.08
Tue Sep 29 13:16:29 2020 TCP/UDP: Preserving recently used remote address: [AF_INET]52.151.227.62:1194
Tue Sep 29 13:16:29 2020 UDP link local: (not bound)
Tue Sep 29 13:16:29 2020 UDP link remote: [AF_INET]52.151.227.62:1194
Tue Sep 29 13:16:29 2020 UDP link local: (not bound)
Tue Sep 29 13:16:29 2020 UDP link remote: [AF_INET]52.151.227.62:1194
Tue Sep 29 13:16:29 2020 NOTE: UID/GID downgrade will be delayed because of --client, --pull, or --up-delay
Tue Sep 29 13:16:31 2020 [openvpn-server.5f7319c92f44fb00017807e0] Peer Connection Initiated with [AF_INET]52.151.227.62:1194
Tue Sep 29 13:16:32 2020 TUN/TAP device tun0 opened
Tue Sep 29 13:16:32 2020 do_ifconfig, tt->did_ifconfig.ipv6_setup=0
Tue Sep 29 13:16:32 2020 /sbin/ip link set dev tun0 up mtu 1500
Tue Sep 29 13:16:32 2020 /sbin/ip addr add dev tun0 192.0.2.2/24 broadcast 192.0.2.255
Tue Sep 29 13:16:32 2020 GID set to nogroup
Tue Sep 29 13:16:32 2020 UID set to nobody
Tue Sep 29 13:16:32 2020 WARNING: this configuration may cache passwords in memory -- use the auth-nocache option to prevent this
Tue Sep 29 13:16:32 2020 Initialization Sequence Completed
```

You can also retrieve those logs by searching the container logs in the logging and monitoring service. This example searches [Azure Monitor container insights](#) to check for **aks-link** connectivity errors.

```
let ContainerIDs = KubePodInventory
| where ClusterId =~
"/subscriptions/YOUR_SUBSCRIPTION_ID/resourceGroups/RESOURCE_GROUP/providers/Microsoft.ContainerService/managedClusters/YOUR_CLUSTER_ID"
| where Name has "aks-link"
| distinct ContainerID;
ContainerLog
| where ContainerID in (ContainerIDs)
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
| order by TimeGenerated desc
| limit 200
```

Here's another example query to check for **tunneelfront** connectivity errors.

```
let ContainerIDs = KubePodInventory
| where ClusterId =~
"/subscriptions/YOUR_SUBSCRIPTION_ID/resourceGroups/RESOURCE_GROUP/providers/Microsoft.ContainerService/managedClusters/YOUR_CLUSTER_ID"
| where Name has "tunneelfront"
| distinct ContainerID;
ContainerLog
| where ContainerID in (ContainerIDs)
| where LogEntry has "ssh to tunnelend is not connected"
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
| order by TimeGenerated desc
| limit 200
```

If you can't get the logs through the kubectl or queries, use [SSH into the node](#). This example finds the **tunneelfront** pod after connecting to the node through SSH.

```
kubectl pods -n kube-system -o wide | grep tunneelfront
ssh azureuser@<agent node pod is on, output from step above>
docker ps | grep tunneelfront
docker logs ...
nslookup <ssh-server_fqdn>
ssh -vv azureuser@<ssh-server_fqdn> -p 9000
docker exec -it <tunneelfront_container_id> /bin/bash -c "ping bing.com"
kubectl get pods -n kube-system -o wide | grep <agent_node_where_tunneelfront_is_running>
kubectl delete po <kube_proxy_pod> -n kube-system
```

3- Validate DNS resolution when restricting egress

DNS resolution is a critical component of your cluster. If DNS resolution isn't working, then control plane errors

or container image pull failures may occur.

Tools:

- nslookup
- dig

Follow these steps to make sure that DNS resolution is working.

1. Exec into the pod to examine and use nslookup or dig if those tools are installed on the pod.
2. If the pod doesn't have those tools, start a utility pod in the same namespace and retry with the tools.
3. If those steps don't show insights, SSH to one of the nodes and try resolution from there. This step will help determine if the issue is related to AKS related or networking configuration.
4. If DNS resolves from the node, then the issue is related to Kubernetes DNS and not a networking issue. Restart Kubernetes DNS and check whether the issue is resolved. If not, open a Microsoft support ticket.
5. If DNS doesn't resolve from the node, then check the networking setup again to make sure that the appropriate routing paths and ports are open.

4- Check for kubelet errors

Check the kubelet process running on each worker node and make sure it's not experiencing any pressures. Those pressures can be related to CPU, memory, or storage.

Tools:

- AKS - Kubelet Workbook

The screenshot shows the AKS - Kubelet Workbook interface. At the top, there is a navigation bar with a search bar, a 'New' button, an 'Open' button, a 'Refresh' button, a smiley face icon, a help icon, and a community icon. Below the navigation bar, a message says 'To get started, choose a report or template below, or use 'Open' to open an existing report.' On the left, there is a sidebar with sections for 'Settings' and 'Monitoring'. Under 'Settings', there are icons for Node pools, Configuration, Scale, Networking, Dev Spaces, Deployment center (preview), Policies, Properties, Locks, and Workbooks (which is highlighted). Under 'Monitoring', there are icons for Insights, Alerts, Metrics, Diagnostic settings, Advisor recommendations, Logs, and Automation. The main area displays a list of 'Kubernetes Services (8)'. Each service is represented by a card with an icon and a name: Deployments & HPAs, Node Disk Capacity, Kubelet (which has an orange border around its card), Node Network, Container Insights Usage, and Workload Details.

- Prometheus and Grafana Dashboard: Kubelet Dashboard



The pressure increases when kubelet restarts and causes some sporadic, unpredictable behavior. Make sure that the error count isn't continuously growing. An occasional error is acceptable but a constant growth indicates an underlying issue that needs to be investigated and resolved.

5- Check disk IOPS for throttling

Check to see that file operations (IOPS) are not getting throttled and impacting services in the cluster.

Tools:

- Azure Monitor container insights Disk IO Workbook

The screenshot shows the Azure portal interface with the following details:

- Search Bar:** Search (Cmd+ /) and a search input field.
- Header:** New, Open, Refresh, Help, Community.
- Left Navigation:**
 - Settings:** Node pools, Configuration, Scale, Networking, Dev Spaces, Deployment center (preview), Policies, Properties, Locks.
 - Monitoring:** Insights, Alerts, Metrics, Diagnostic settings, Advisor recommendations, Logs, Workbooks (highlighted).
 - Automation:**
- Middle Content:** A message: "To get started, choose a report or template below, or use 'Open' to open an existing report." Below it is a search bar.
- Kubernetes Services:** Recently modified container insights reports (0) - No items found.
- Report Cards:** A grid of cards:
 - Deployments & HPAs
 - Node Disk Capacity
 - Node Disk IO** (highlighted)
 - Node Network
 - Node GPU
 - Container Insights Usage
 - Workload

- Prometheus and Grafana Dashboard: Node Disk Dashboard



Physical storage devices have limitations, bandwidth, and total number of file operations. Azure Disks are used to store the OS running on the AKS nodes. They are subject to the same physical storage limitations.

Another way is to look at the throughput. IOPSs is measured with the average IO size * IOPS as the throughput in MB/s. Large IO sizes will lead to lower IOPS because the throughput of a disk doesn't change.

When a workload exceeds Azure Disks' service limits on Max IOPS, the cluster becomes unresponsive and blocked in IO Wait. Everything on Linux is a file. This includes network sockets, CNI, Docker, and other services that use network I/O. All of those files will fail if they're unable to read the disk.

The events that can trigger IOPS throttle include:

- High volume of containers running on the nodes. Docker IO is shared on the OS disk.
- Custom or third-party tools used for security, monitoring, logging that are running on the OS disk.
- Node failover events, and periodic jobs. As the load increases or the pods are scaled, this throttling occurs more frequently until all nodes go to **NotReady** state while the IO completes.

Related links

[Virtual machine disk limits](#)

[Relationship between Virtual Machine and Disk Performance](#)

Next steps

[Check the workload deployments](#)

Check the workload deployments

3/10/2022 • 2 minutes to read • [Edit Online](#)

Check to see that all deployments and daemonSets are running. The **Ready** and the **Available** matches the expected count.

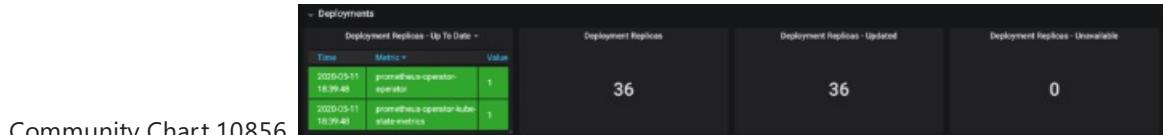
This article is part of a series. Read the introduction [here](#).

Tools:

- **AKS - Workloads.** In Azure portal, navigate to the AKS cluster resource. Select **Workloads**.

Name	Namespace	Ready	Up-to-date	Available	Age
coredns	kube-system	1/1	2	2	3 hours
coredns-autoscaler	kube-system	1/1	1	1	3 hours
metrics-server	kube-system	1/1	1	1	3 hours
omsagent-rs	kube-system	1/1	1	1	3 hours
azure-policy	kube-system	1/1	1	1	3 hours
azure-policy-webhook	kube-system	1/1	1	1	3 hours
calico-typa	kube-system	1/1	1	1	3 hours

- **Prometheus and Grafana Dashboard.** Deployment Status Dashboard. This image is from Grafana



Next steps

[Validate the admission controllers](#)

Validate the admission controllers are working as expected

3/10/2022 • 2 minutes to read • [Edit Online](#)

Check whether the admission controllers are working as expected.

This article is part of a series. Read the introduction [here](#).

Issues because of admission controllers are an edge case but they should be considered. Here are some examples:

- Mutating and validating webhooks. Be careful when you add mutating and validating webhooks in your cluster. Make sure that they're highly available so that an unhealthy node doesn't cause API server requests to be blocked. AKS Policy, also known as OPA Gatekeeper, is an example of this type of webhooks. If there are problems in the admission control pipeline, it can block a large number of requests to the API server.
- Service meshes. They use admission controllers to automatically inject sidecars for example.

Tools `kubectl`

These commands check if AKS Policy is running in your cluster and how to validate that all of the admission controllers are functioning as expected.

```
# Check AKS Policy pods are running.  
kubectl get po -n gatekeeper-system  
  
# Sample Output  
...  
NAME READY STATUS RESTARTS AGE  
gatekeeper-audit-65844778cb-rkf1g 1/1 Running 0 163m  
gatekeeper-controller-78797d4687-4pf6w 1/1 Running 0 163m  
gatekeeper-controller-78797d4687-splzh 1/1 Running 0 163m  
...
```

If this command doesn't run as expected, it could indicate that an admission controller, API service, or CRD isn't functioning correctly.

```
# Check that all API Resources are working correctly.  
kubectl api-resources  
  
# Sample Output  
...  
NAME SHORTNAMES APIGROUP NAMESPACED KIND  
bindings cs true Binding  
componentstatuses cs false ComponentStatus  
configmaps cm true ConfigMap  
...
```

Next steps

[Verify the connection to the container registry](#)

Verify the connection to the container registry

3/10/2022 • 2 minutes to read • [Edit Online](#)

Make sure that the worker nodes have the correct permission to pull the necessary container images from the container registry.

This article is part of a series. Read the introduction [here](#).

A common symptom of this issue is receiving **ImagePullBackoff** errors when getting or describing a pod. Be sure that the registry and image name are correct. Also, the cluster has permissions to pull from the appropriate container registry.

If you are using Azure Container Registry (ACR), the cluster service principal or managed identity should be granted **AcrPull** permissions against the registry.

One way is to run this command using the managed identity of the AKS cluster node pool. This command gets a list of its permissions.

```
# Get Kubelet Identity (Nodepool MSI)
ASSIGNEE=$(az aks show -g $RESOURCE_GROUP -n $NAME --query identityProfile.kubeletidentity.clientId -o tsv)
az role assignment list --assignee $ASSIGNEE --all -o table
```

```
# Expected Output
...
e5615a90-1767-4a4f-83b6-cecfa0675970  AcrPull
/subscriptions/.../providers/Microsoft.ContainerRegistry/registries/akskhacr
...
```

If you're using another container registry, check the appropriate **ImagePullSecret** credentials for the registry.

Related links

[Import container images to a container registry](#)

[AKS Roadmap](#)

Patching and upgrade guidance

3/10/2022 • 5 minutes to read • [Edit Online](#)

This section of the Azure Kubernetes Service (AKS) day-2 operations guide describes patching and upgrading practices for AKS worker nodes and Kubernetes (K8S) versions.

Node image upgrades

Microsoft provides patches and new images for image nodes weekly, but doesn't automatically patch them by default. Because AKS isn't a platform-as-a-service (PaaS), components like agent nodes have shared responsibility, and users must help maintain the AKS cluster. For example, applying an agent node operating system (OS) security patch requires user input.

AKS supports upgrading node images by using [az aks nodepool upgrade](#), so you can keep up with the newest OS and runtime updates. To keep your agent node OS and runtime components patched, consider checking and applying node image upgrades bi-weekly, or automating the node image upgrade process. For more information about automating node image upgrades, see [Node upgrade GitHub Actions](#).

An updated node image contains up-to-date OS security patches, kernel updates, Kubernetes security updates, newer versions of binaries like `kubelet`, and component version updates listed in the [release notes](#). Node image updates have all relevant and validated security updates and feature updates. Using the node image upgrade method ensures you get only tested kernels and components that are compatible with those kernels.

You can use node image upgrades to streamline upgrades for both Windows and Linux node pools, but the processes differ slightly. Linux might receive daily security updates, but Windows Server nodes update by performing an AKS upgrade that deploys new nodes with the latest base Window Server image and patches.

Use [kubectl describe nodes](#) to check the OS kernel version and the OS image version of the nodes in your cluster:

```
kubectl describe nodes <NodeName>
```

Example output:

```
System Info:  
Machine ID: 12345678-1234-1234-0123456789ab  
System UUID: abcdefga-abcd-abcd-abcd-abcdefg01234  
Boot ID: abcd0123-ab01-01ab-ab01-abcd01234567  
Kernel Version: 4.15.0-1096-azure  
OS Image: Ubuntu 16.04.7 LTS  
Operating System: linux  
Architecture: amd64  
Container Runtime Version: docker://19.3.12  
Kubelet Version: v1.17.9  
Kube-Proxy Version: v1.17.9
```

Use the Azure CLI [az aks nodepool list](#) command to check the current node image versions of the nodes in a cluster:

```
az aks nodepool list \
--resource-group <ResourceGroupName> --cluster-name <AKSClusterName> \
--query "[].{Name:name,NodeImageVersion:nodeImageVersion}" --output table
```

Example output:

Name	NodeImageVersion
systempool	AKSUbuntu-1604-2020.09.30
usernodepool	AKSUbuntu-1604-2020.09.30
usernp179	AKSUbuntu-1604-2020.10.28

Use [az aks nodepool get-upgrades](#) to find out the latest available node image version:

```
az aks nodepool get-upgrades \
--resource-group <ResourceGroupName> --cluster-name <AKSClusterName> \
--nodepool-name <NodePoolName> --output table
```

Example output:

KubernetesVersion	LatestNodeImageVersion	Name	OsType
1.16.13	AKSUbuntu-1604-2020.11.11	default	Linux

To upgrade node pools to the latest node image version:

- [Upgrade all nodes in node pools.](#)
- [Upgrade a specific node pool.](#)
- [Automate node pool upgrades using GitHub Actions.](#)
- [Automate node pool upgrades using auto-upgrade channels.](#)

Cluster upgrades

The Kubernetes community releases minor K8S versions roughly every three months. The [AKS GitHub release notes page](#) publishes information about new AKS versions and releases, the latest AKS features, behavioral changes, bug fixes, and component updates. You can also subscribe to the [GitHub AKS RSS feed](#).

The window of supported K8S versions on AKS is called "N - 2": (N (latest release) - 2 (minor versions)). It's important to establish a continuous cluster upgrade process to ensure that your AKS clusters don't go out of support. Once a new version becomes available, ideally you should plan an upgrade across all environments before the version becomes the default. This approach provides more control and predictability, and lets you plan upgrades with minimal disruption to existing workloads.

To minimize disruption to existing workloads during an upgrade:

- Set up multiple environments.
- Plan and schedule maintenance windows.
- [Plan your tolerance for disruption.](#)
- [Use surge upgrades to control node pool upgrades.](#)

To check when your cluster requires an upgrade, use [az aks get-upgrades](#) to get a list of available target upgrade versions for your AKS control plane. Determine the target version for your control plane from the results.

```
az aks get-upgrades \
--resource-group <ResourceGroupName> --name <AKSClusterName> --output table
```

Example output:

MasterVersion	Upgrades
1.17.9	1.17.11, 1.17.13, 1.18.8, 1.18.10

Check the Kubernetes versions of the nodes in your node pools to determine the node pools that need to be upgraded.

```
az aks nodepool list \
--resource-group <ResourceGroupName> --cluster-name <AKSClusterName> \
--query "[].{Name:name,k8version:orchestratorVersion}" --output table
```

Example output:

Name	K8version
systempool	1.16.13
usernodepool	1.16.13
usernp179	1.17.9

You can upgrade the control plane first, and then upgrade the individual node pools.

1. Run the [az aks upgrade](#) command with the `--control-plane-only` flag to upgrade only the cluster control plane, and not any of the associated node pools:

```
az aks upgrade \
--resource-group <ResourceGroupName> --name <AKSClusterName> \
--control-plane-only --no-wait \
--kubernetes-version <KubernetesVersion>
```

2. Run [az aks nodepool upgrade](#) to upgrade node pools to the target version:

```
az aks nodepool upgrade \
--resource-group <ResourceGroupName> --cluster-name <AKSClusterName> --name <NodePoolName> \
--no-wait --kubernetes-version <KubernetesVersion>
```

For information about validation rules for cluster upgrades, see [Validation rules for upgrades](#).

Considerations

The following table describes characteristics of various AKS upgrade and patching scenarios:

SCENARIO	USER INITIATED	K8S UPGRADE	OS KERNEL UPGRADE	NODE IMAGE UPGRADE
Security patching	No	No	Yes, following reboot	Yes

SCENARIO	USER INITIATED	K8S UPGRADE	OS KERNEL UPGRADE	NODE IMAGE UPGRADE
Cluster create	Yes	Maybe	Yes, if an updated node image uses an updated kernel.	Yes, relative to an existing cluster if a new release is available.
Control plane K8S upgrade	Yes	Yes	No	No
Node pool K8S upgrade	Yes	Yes	Yes, if an updated node image uses an updated kernel.	Yes, if a new release is available.
Node pool scale up	Yes	No	No	No
Node image upgrade	Yes	No	Yes, if an updated node image uses an updated kernel.	Yes
Cluster auto upgrade	No	Yes	No	No

- For more information about Linux Automatic Security Updates, see [AutomaticSecurityUpdates](#).
- It's possible that an OS security patch applied as part of a node image upgrade will install a later version of the kernel than creating a new cluster.
- You can use the [Agent Pools - Get Upgrade Profile](#) API to determine the latest node image version.
- Node pool scale up uses the model associated with the virtual machine scale set at creation. OS kernels upgrade when security patches are applied and the nodes reboot.
- Cluster auto upgrade is in preview. For more information, see [Set auto-upgrade channel](#).
- Node image auto upgrade is in development. For more information, see [Automatic Node Image Upgrade for node versions](#).

See also

- [AKS day-2 operations guide](#)
- [AKS triage practices](#)
- [AKS common issues](#)

Related links

- [AKS product documentation](#)
- [AKS Roadmap](#)
- [Defining Day-2 Operations](#)

Choose a Kubernetes at the edge compute option

3/10/2022 • 6 minutes to read • [Edit Online](#)

This document discusses the trade-offs for various options available for extending compute on the edge. The following considerations for each Kubernetes option are covered:

- **Operational cost.** The expected labor required to maintain and operate the Kubernetes clusters.
- **Ease of configuration.** The level of difficulty to configure and deploy a Kubernetes cluster.
- **Flexibility.** A measure of how adaptable the Kubernetes option is to integrate a customized configuration with existing infrastructure at the edge.
- **Mixed node.** Ability to run a Kubernetes cluster with both Linux and Windows nodes.

Assumptions

- You are a cluster operator looking to understand different options for running Kubernetes at the edge and managing clusters in Azure.
- You have a good understanding of existing infrastructure and any other infrastructure requirements, including storage and networking requirements.

After reading this document, you'll be in a better position to identify which option best fits your scenario and the environment required.

Kubernetes choices at a glance

	OPERATIONAL COST	EASE OF CONFIGURATION	FLEXIBILITY	MIXED NODE	SUMMARY
Bare-metal Kubernetes	High**	Difficult**	High**	Yes	A ground-up configuration on any available infrastructure at location with the option to use Azure Arc for added Azure capabilities.
K8s on Azure Stack Edge Pro	Low	Easy	Low	Linux only	Kubernetes deployed on Azure Stack Edge appliance deployed at location.
AKS on HCI	Low	Easy	Medium	Yes	AKS deployed on Azure Stack HCI or Windows Server 2019.

*Other managed edge platforms (OpenShift, Tanzu, and so on) aren't in scope for this document.

**These values are based on using `kubeadm`, for the sake of simplicity. Different options for running bare-metal Kubernetes at the edge would alter the rating in these categories.

Bare-metal Kubernetes

Ground-up configuration of Kubernetes using tools like `kubeadm` on any underlying infrastructure.

The biggest constraints for bare-metal Kubernetes are around the specific needs and requirements of the organization. The opportunity to use any distribution, networking interface, and plugin means higher complexity and operational cost. But this offers the most flexible option for customizing your cluster.

Scenario

Often, *edge* locations have specific requirements for running Kubernetes clusters that aren't met with the other Azure solutions described in this document. Meaning this option is typically best for those unable to use managed services due to unsupported existing infrastructure, or those who seek to have maximum control of their clusters.

- This option can be especially difficult for those who are new to Kubernetes. This isn't uncommon for organizations looking to run edge clusters. Options like [MicroK8s](#) or [k3s](#) aim to flatten that learning curve.
- It's important to understand any underlying infrastructure and any integration that is expected to take place up front. This will help to narrow down viable options and to identify any gaps with the open-source tooling and/or plugins.
- Enabling clusters with [Azure Arc](#) presents a simple way to manage your cluster from Azure alongside other resources. This also brings other Azure capabilities to your cluster, including [Azure Policy](#), [Azure Monitor](#), [Microsoft Defender for Cloud](#), and other services.
- Because cluster configuration isn't trivial, it's especially important to be mindful of CI/CD. Tracking and acting on upstream changes of various plugins, and making sure those changes don't affect the health of your cluster, becomes a direct responsibility. It's important for you to have a strong CI/CD solution, strong testing, and monitoring in place.

Tooling options

Cluster bootstrap:

- [kubeadm](#): Kubernetes tool for creating ground-up Kubernetes clusters. Good for standard compute resources (Linux/Windows).
- [MicroK8s](#): Simplified administration and configuration ("LowOps"), conformant Kubernetes by Canonical.
- [k3s](#): Certified Kubernetes distribution built for Internet of Things (IoT) and edge computing.

Storage:

- Explore available [CSI drivers](#): Many options are available to fit your requirements from cloud to local file shares.

Networking:

- A full list of available add-ons can be found here: [Networking add-ons](#). Some popular options include [Flannel](#), a simple overlay network, and [Calico](#), which provides a full networking stack.

Considerations

Operational cost:

- Without the support that comes with managed services, it's up to the organization to maintain and operate

the cluster as a whole (storage, networking, upgrades, observability, application management). The operational cost is considered high.

Ease of configuration:

- Evaluating the many open-source options at every stage of configuration whether its networking, storage, or monitoring options is inevitable and can become complex. Requires more consideration for configuring a CI/CD for cluster configuration. Because of these concerns, the ease of configuration is considered difficult.

Flexibility:

- With the ability to use any open-source tool or plugin without any provider restrictions, bare-metal Kubernetes is highly flexible.

Kubernetes on Azure Stack Edge

Kubernetes cluster (a master VM and a worker VM) configured and deployed for you on your Azure Stack Edge Pro device.

[Azure Stack Edge Pro](#) devices deliver Azure capabilities like compute, storage, networking, and hardware-accelerated machine learning (ML) to any edge location. Kubernetes clusters can be created once the compute role is enabled on any of the Pro-GPU, Pro-R, and Mini-R devices. Managing upgrades of the Kubernetes cluster can be done using standard updates available for the device.

Scenario

Ideal for those with existing (Linux) IoT workloads or upgrading their compute for ML at the edge. This is a good option when it isn't necessary to have more granular control over the clusters.

- Admin permissions aren't granted by default. Although you can work with the product group to make certain exceptions, this makes it difficult to have finer control of your cluster.
- There is an extra [cost](#) if there isn't already an Azure Stack Edge device. Explore [Azure Stack Edge devices](#) and see if any fit your compute requirements.
- [Calico](#), [MetallLB](#), and [CoreDNS](#) are installed for Kubernetes networking on the device.
- Only [Linux](#) workloads are supported at this time.
- In addition to Kubernetes, Azure Stack Edge also comes with the IoT runtime, which means that workloads may also be deployed to your Azure Stack Edge clusters via IoT Edge.
- Support for two node clusters isn't currently available. This effectively means that this option is *not* a highly available (HA) solution.

Considerations

Operational cost:

- With the support that comes with the device, operational cost is minimal and is scoped to workload management.

Ease of configuration:

- Pre-configured and well-documented Kubernetes cluster deployment simplifies the configuration required compared to bare-metal Kubernetes.

Flexibility:

- Configuration is already set, and Admin permissions aren't granted by default. Product group involvement may be required beyond basic configuration, and the underlying infrastructure must be an Azure Stack Edge

Pro device, making this a less flexible option.

AKS on HCI

Note: This option is currently in [preview](#).

AKS-HCI is a set of predefined settings and configurations that is used to deploy one or more Kubernetes clusters (with Windows Admin Center or PowerShell modules) on a multi-node cluster running either Windows Server 2019 Datacenter or Azure Stack HCI 20H2.

Scenario

Ideal for those who want a simplified and streamlined way to get a Microsoft-supported cluster on compatible devices (Azure Stack HCI or Windows Server 2019 Datacenter). Operations and configuration complexity are reduced at the expense of the flexibility when compared to the bare-metal Kubernetes option.

Considerations

At the time of this writing, the preview comes with many limitations (permissions, networking limitations, large compute requirements, and documentation gaps). Purposes other than evaluation and development are discouraged at this time.

Operational cost:

- Microsoft-supported cluster minimizes operational costs.

Ease of configuration:

- Pre-configured and well-documented Kubernetes cluster deployment simplifies the configuration required compared to bare-metal Kubernetes.

Flexibility:

- Cluster configuration itself is set, but Admin permissions are granted. The underlying infrastructure must either be Azure Stack HCI or Windows Server 2019. This option is more flexible than Kubernetes on Azure Stack Edge and less flexible than bare-metal Kubernetes.

Next steps

For more information, see the following articles:

- [What is Azure IoT Edge](#)
- [Kubernetes on your Azure Stack Edge Pro GPU device](#)
- [Use IoT Edge module to run a Kubernetes stateless application on your Azure Stack Edge Pro GPU device](#)
- [Deploy a Kubernetes stateless application via kubectl on your Azure Stack Edge Pro GPU device](#)
- [AI at the edge with Azure Stack Hub](#)
- [Building a CI/CD pipeline for microservices on Kubernetes](#)
- [Use Kubernetes dashboard to monitor your Azure Stack Edge Pro GPU device](#)

Azure Data Architecture Guide

3/10/2022 • 2 minutes to read • [Edit Online](#)

This guide presents a structured approach for designing data-centric solutions on Microsoft Azure. It is based on proven practices derived from customer engagements.

NOTE

Learn more about adopting your systems for data governance, analytics, and data management, in [Cloud adoption for data management](#).

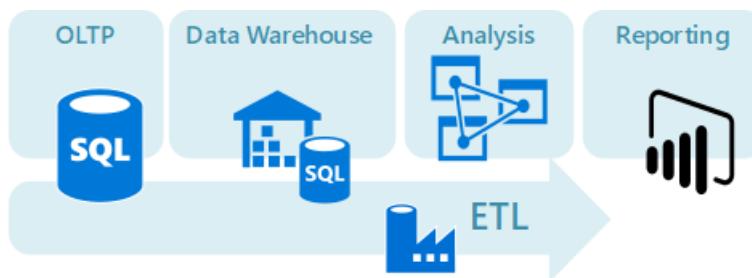
Introduction

The cloud is changing the way applications are designed, including how data is processed and stored. Instead of a single general-purpose database that handles all of a solution's data, *Polyglot persistence* solutions use multiple, specialized data stores, each optimized to provide specific capabilities. The perspective on data in the solution changes as a result. There are no longer multiple layers of business logic that read and write to a single data layer. Instead, solutions are designed around a *data pipeline* that describes how data flows through a solution, where it is processed, where it is stored, and how it is consumed by the next component in the pipeline.

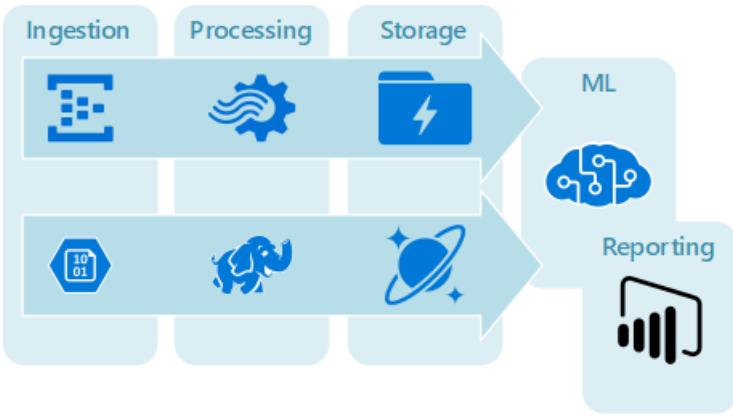
How this guide is structured

This guide is structured around two general categories of data solution, *traditional RDBMS workloads* and *big data solutions*.

Traditional RDBMS workloads. These workloads include online transaction processing (OLTP) and online analytical processing (OLAP). Data in OLTP systems is typically relational data with a predefined schema and a set of constraints to maintain referential integrity. Often, data from multiple sources in the organization may be consolidated into a data warehouse, using an ETL process to move and transform the source data.



Big data solutions. A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems. The data may be processed in batch or in real time. Big data solutions typically involve a large amount of non-relational data, such as key-value data, JSON documents, or time series data. Often traditional RDBMS systems are not well-suited to store this type of data. The term *NoSQL* refers to a family of databases designed to hold non-relational data. The term isn't quite accurate, because many non-relational data stores support SQL compatible queries. The term *NoSQL* stands for "Not only SQL".



These two categories are not mutually exclusive, and there is overlap between them, but we feel that it's a useful way to frame the discussion. Within each category, the guide discusses **common scenarios**, including relevant Azure services and the appropriate architecture for the scenario. In addition, the guide compares **technology choices** for data solutions in Azure, including open source options. Within each category, we describe the key selection criteria and a capability matrix, to help you choose the right technology for your scenario.

This guide is not intended to teach you data science or database theory — you can find entire books on those subjects. Instead, the goal is to help you select the right data architecture or data pipeline for your scenario, and then select the Azure services and technologies that best fit your requirements. If you already have an architecture in mind, you can skip directly to the technology choices.

Online transaction processing (OLTP)

3/10/2022 • 6 minutes to read • [Edit Online](#)

The management of transactional data using computer systems is referred to as online transaction processing (OLTP). OLTP systems record business interactions as they occur in the day-to-day operation of the organization, and support querying of this data to make inferences.

Transactional data

Transactional data is information that tracks the interactions related to an organization's activities. These interactions are typically business transactions, such as payments received from customers, payments made to suppliers, products moving through inventory, orders taken, or services delivered. Transactional events, which represent the transactions themselves, typically contain a time dimension, some numerical values, and references to other data.

Transactions typically need to be *atomic* and *consistent*. Atomicity means that an entire transaction always succeeds or fails as one unit of work, and is never left in a half-completed state. If a transaction cannot be completed, the database system must roll back any steps that were already done as part of that transaction. In a traditional RDBMS, this rollback happens automatically if a transaction cannot be completed. Consistency means that transactions always leave the data in a valid state. (These are very informal descriptions of atomicity and consistency. There are more formal definitions of these properties, such as [ACID](#).)

Transactional databases can support strong consistency for transactions using various locking strategies, such as pessimistic locking, to ensure that all data is strongly consistent within the context of the enterprise, for all users and processes.

The most common deployment architecture that uses transactional data is the data store tier in a 3-tier architecture. A 3-tier architecture typically consists of a presentation tier, business logic tier, and data store tier. A related deployment architecture is the [N-tier](#) architecture, which may have multiple middle-tiers handling business logic.

Typical traits of transactional data

Transactional data tends to have the following traits:

REQUIREMENT	DESCRIPTION
Normalization	Highly normalized
Schema	Schema on write, strongly enforced
Consistency	Strong consistency, ACID guarantees
Integrity	High integrity
Uses transactions	Yes
Locking strategy	Pessimistic or optimistic
Updateable	Yes

Requirement	Description
Appendable	Yes
Workload	Heavy writes, moderate reads
Indexing	Primary and secondary indexes
Datum size	Small to medium sized
Model	Relational
Data shape	Tabular
Query flexibility	Highly flexible
Scale	Small (MBs) to Large (a few TBs)

When to use this solution

Choose OLTP when you need to efficiently process and store business transactions and immediately make them available to client applications in a consistent way. Use this architecture when any tangible delay in processing would have a negative impact on the day-to-day operations of the business.

OLTP systems are designed to efficiently process and store transactions, as well as query transactional data. The goal of efficiently processing and storing individual transactions by an OLTP system is partly accomplished by data normalization — that is, breaking the data up into smaller chunks that are less redundant. This supports efficiency because it enables the OLTP system to process large numbers of transactions independently, and avoids extra processing needed to maintain data integrity in the presence of redundant data.

Challenges

Implementing and using an OLTP system can create a few challenges:

- OLTP systems are not always good for handling aggregates over large amounts of data, although there are exceptions, such as a well-planned SQL Server-based solution. Analytics against the data, that rely on aggregate calculations over millions of individual transactions, are very resource intensive for an OLTP system. They can be slow to execute and can cause a slow-down by blocking other transactions in the database.
- When conducting analytics and reporting on data that is highly normalized, the queries tend to be complex, because most queries need to de-normalize the data by using joins. Also, naming conventions for database objects in OLTP systems tend to be terse and succinct. The increased normalization coupled with terse naming conventions makes OLTP systems difficult for business users to query, without the help of a DBA or data developer.
- Storing the history of transactions indefinitely and storing too much data in any one table can lead to slow query performance, depending on the number of transactions stored. The common solution is to maintain a relevant window of time (such as the current fiscal year) in the OLTP system and offload historical data to other systems, such as a data mart or [data warehouse](#).

OLTP in Azure

Applications such as websites hosted in [App Service Web Apps](#), REST APIs running in App Service, or mobile or desktop applications communicate with the OLTP system, typically via a REST API intermediary.

In practice, most workloads are not purely OLTP. There tends to be an analytical component as well. In addition, there is an increasing demand for real-time reporting, such as running reports against the operational system. This is also referred to as HTAP (Hybrid Transactional and Analytical Processing). For more information, see [Online Analytical Processing \(OLAP\)](#).

In Azure, all of the following data stores will meet the core requirements for OLTP and the management of transaction data:

- [Azure SQL Database](#)
- [SQL Server in an Azure virtual machine](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Does your solution have specific dependencies for Microsoft SQL Server, MySQL or PostgreSQL compatibility? Your application may limit the data stores you can choose based on the drivers it supports for communicating with the data store, or the assumptions it makes about which database is used.
- Are your write throughput requirements particularly high? If yes, choose an option that provides in-memory tables.
- Is your solution multitenant? If so, consider options that support capacity pools, where multiple database instances draw from an elastic pool of resources, instead of fixed resources per database. This can help you better distribute capacity across all database instances, and can make your solution more cost effective.
- Does your data need to be readable with low latency in multiple regions? If yes, choose an option that supports readable secondary replicas.
- Does your database need to be highly available across geo-graphic regions? If yes, choose an option that supports geographic replication. Also consider the options that support automatic failover from the primary replica to a secondary replica.
- Does your database have specific security needs? If yes, examine the options that provide capabilities like row level security, data masking, and transparent data encryption.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

ABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Is Managed Service	Yes	No	Yes	Yes
Runs on Platform	N/A	Windows, Linux, Docker	N/A	N/A

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Programmability ¹	T-SQL, .NET, R	T-SQL, .NET, R, Python	SQL	SQL, PL/pgSQL

[1] Not including client driver support, which allows many programming languages to connect to and use the OLTP data store.

Scalability capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Maximum database instance size	4 TB	256 TB	16 TB	16 TB
Supports capacity pools	Yes	Yes	No	No
Supports clusters scale out	No	Yes	No	No
Dynamic scalability (scale up)	Yes	No	Yes	Yes

Analytic workload capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Temporal tables	Yes	Yes	No	No
In-memory (memory-optimized) tables	Yes	Yes	No	No
Columnstore support	Yes	Yes	No	No
Adaptive query processing	Yes	Yes	No	No

Availability capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Readable secondaries	Yes	Yes	Yes	Yes
Geographic replication	Yes	Yes	Yes	Yes
Automatic failover to secondary	Yes	No	No	No

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Point-in-time restore	Yes	Yes	Yes	Yes

Security capabilities

CAPABILITY	AZURE SQL DATABASE	SQL SERVER IN AN AZURE VIRTUAL MACHINE	AZURE DATABASE FOR MYSQL	AZURE DATABASE FOR POSTGRESQL
Row level security	Yes	Yes	Yes	Yes
Data masking	Yes	Yes	No	No
Transparent data encryption	Yes	Yes	Yes	Yes
Restrict access to specific IP addresses	Yes	Yes	Yes	Yes
Restrict access to allow VNet access only	Yes	Yes	Yes	Yes
Azure Active Directory authentication	Yes	No	Yes	Yes
Active Directory authentication	No	Yes	No	No
Multi-factor authentication	Yes	No	Yes	Yes
Supports Always Encrypted	Yes	Yes	No	No
Private IP	No	Yes	No	No

Working with CSV and JSON files for data solutions

3/10/2022 • 4 minutes to read • [Edit Online](#)

CSV and JSON are likely the most common formats used for ingesting, exchanging, and storing unstructured or semi-structured data.

About CSV format

CSV (comma-separated values) files are commonly used to exchange tabular data between systems in plain text. They typically contain a header row that provides column names for the data, but are otherwise considered semi-structured. This is due to the fact that CSVs cannot naturally represent hierarchical or relational data. Data relationships are typically handled with multiple CSV files, where foreign keys are stored in columns of one or more files, but the relationships between those files are not expressed by the format itself. Files in CSV format may use other delimiters besides commas, such as tabs or spaces.

Despite their limitations, CSV files are a popular choice for data exchange, because they are supported by a wide range of business, consumer, and scientific applications. For example, database and spreadsheet programs can import and export CSV files. Similarly, most batch and stream data processing engines, such as Spark and Hadoop, natively support serializing and deserializing CSV-formatted files and offer ways to apply a schema on read. This makes it easier to work with the data, by offering options to query against it and store the information in a more efficient data format for faster processing.

About JSON format

JSON (JavaScript Object Notation) data is represented as key-value pairs in a semi-structured format. JSON is often compared to XML, as both are capable of storing data in hierarchical format, with child data represented inline with its parent. Both are self-describing and human readable, but JSON documents tend to be much smaller, leading to their popular use in online data exchange, especially with the advent of REST-based web services.

JSON-formatted files have several benefits over CSV:

- JSON maintains hierarchical structures, making it easier to hold related data in a single document and represent complex relationships.
- Most programming languages provide native support for deserializing JSON into objects, or provide lightweight JSON serialization libraries.
- JSON supports lists of objects, helping to avoid messy translations of lists into a relational data model.
- JSON is a commonly used file format for NoSQL databases, such as MongoDB, Couchbase, and Azure Cosmos DB.

Since a lot of data coming across the wire is already in JSON format, most web-based programming languages support working with JSON natively, or through the use of external libraries to serialize and deserialize JSON data. This universal support for JSON has led to its use in logical formats through data structure representation, exchange formats for hot data, and data storage for cold data.

Many batch and stream data processing engines natively support JSON serialization and deserialization. Though the data contained within JSON documents may ultimately be stored in a more performance-optimized formats, such as Parquet or Avro, it serves as the raw data for source of truth, which is critical for reprocessing the data as needed.

When to use CSV or JSON formats

CSVs are more commonly used for exporting and importing data, or processing it for analytics and machine learning. JSON-formatted files have the same benefits, but are more common in hot data exchange solutions. JSON documents are often sent by web and mobile devices performing online transactions, by IoT (internet of things) devices for one-way or bidirectional communication, or by client applications communicating with SaaS and PaaS services or serverless architectures.

CSV and JSON file formats both make it easy to exchange data between dissimilar systems or devices. Their semi-structured formats allow flexibility in transferring almost any type of data, and universal support for these formats make them simple to work with. Both can be used as the raw source of truth in cases where the processed data is stored in binary formats for more efficient querying.

Working with CSV and JSON data in Azure

Azure provides several solutions for working with CSV and JSON files, depending on your needs. The primary landing place for these files is either Azure Storage or Azure Data Lake Store. Most Azure services that work with these and other text-based files integrate with either object storage service. In some situations, however, you may opt to directly import the data into Azure SQL or some other data store. SQL Server has native support for storing and working with JSON documents, which makes it easy to [import and process those types of files](#). You can use a utility like SQL Bulk Import to easily [import CSV files](#).

You can also query JSON files directly from Azure Blob Storage without importing them into Azure SQL. For a complete example of this approach, see [Work with JSON files with Azure SQL](#). Currently this option isn't available for CSV files.

Depending on the scenario, you may perform [batch processing](#) or [real-time processing](#) of the data.

Challenges

There are some challenges to consider when working with these formats:

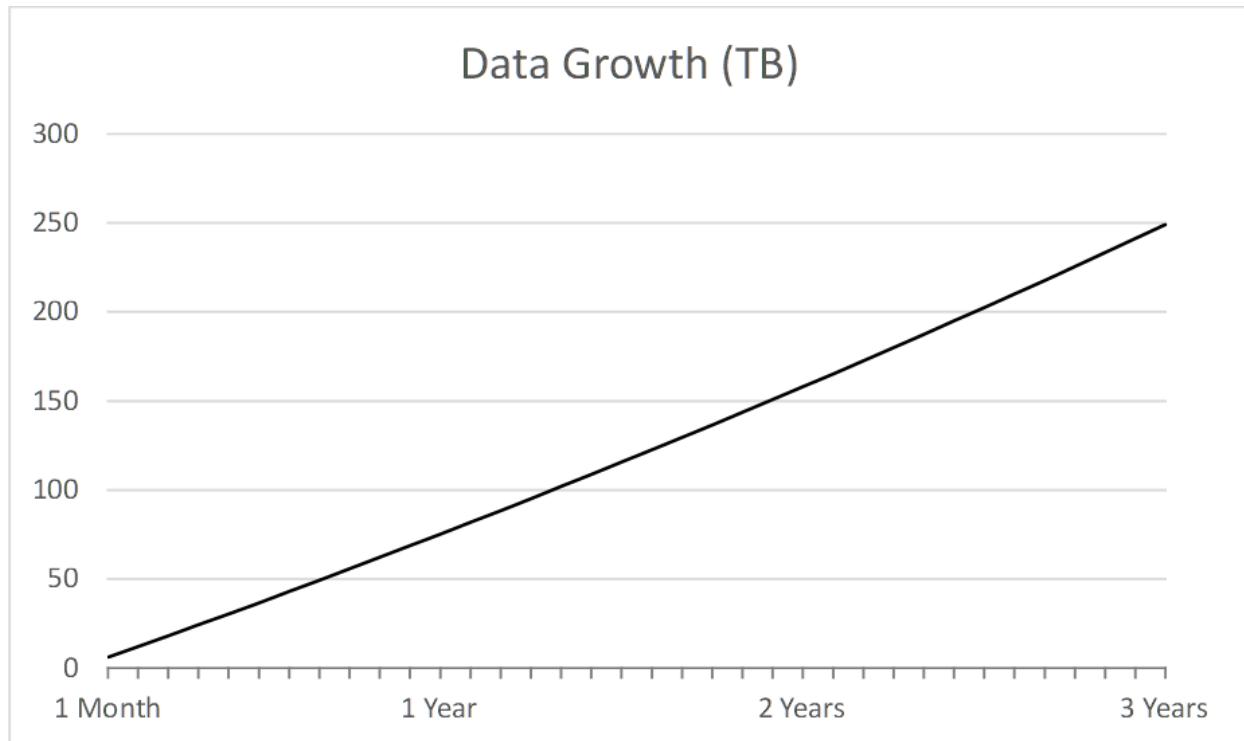
- Without any restraints on the data model, CSV and JSON files are prone to data corruption ("garbage in, garbage out"). For instance, there's no notion of a date/time object in either file, so the file format does not prevent you from inserting "ABC123" in a date field, for example.
- Using CSV and JSON files as your cold storage solution does not scale well when working with big data. In most cases, they cannot be split into partitions for parallel processing, and cannot be compressed as well as binary formats. This often leads to processing and storing this data into read-optimized formats such as Parquet and ORC (optimized row columnar), which also provide indexes and inline statistics about the data contained.
- You may need to apply a schema on the semi-structured data to make it easier to query and analyze. Typically, this requires storing the data in another form that complies with your environment's data storage needs, such as within a database.

Build a scalable system for massive data

3/10/2022 • 2 minutes to read • [Edit Online](#)

Your data storage system is fundamental to the success of your applications, and therefore to the success of your enterprise. When the storage system is well architected, response is quick, data storage capacity is easily adjusted as necessary, the system is resilient to failures, and it's affordable.

A crucial consideration is whether the design scales well as data grows. As an example of data growth, consider an application that generates 6 terabytes (TB) of data its first month, with the amount increasing every month at a 10 percent yearly rate. Here's a graph that shows how data accumulates over time:



After three years, there's 249 TB of data. If the system is well architected, it handles such data growth gracefully, remaining responsive, resilient, and affordable.

This example isn't extreme. If your customers are businesses, data grows both as you add customers and as your customers add data. It can also grow because of application enhancements.

Handling data growth may require a mix of storage products. For example, you may need to keep rarely accessed data in low-cost services, and frequently accessed data in higher-cost services with better access times.

To design such a system on Azure, you need to be familiar with the many Azure services and with how to use them for various types of applications and various objectives. The articles in this section provide seven system architectures for web applications that use massive amounts of data and that are resilient to system failures. They serve as examples that can help you design a storage system that properly accommodates your applications.

The architectures demonstrate the use of these Azure products: Azure Table Storage, Azure Cosmos DB, Azure Data Factory, and Azure Data Lake.

This capability matrix provides links to the articles and summarizes the benefits and risks of each architecture:

ARCHITECTURE	BENEFITS	RISKS
Two-region web application with Table Storage failover	Straightforward, low-cost implementation	Limited resiliency—only two Azure regions
Multi-region web application with custom Storage Table replication	Resiliency	Implementation time and difficulty
Multi-region web application with Cosmos DB replication	Resiliency, performance, scalability	Storage costs
Optimized storage with logical data classification	Resiliency, performance, scalability, storage costs	Implementation time, need to design logical data classification
Optimized Storage – time based – multi writes	Storage costs	Limited resiliency, performance, limited scalability, implementation time, need to design time-based data retention
Optimized Storage – time based with Data Lake	Resiliency, performance, scalability	Implementation time, need to design time-based data retention
Minimal storage – change feed to replicate data	Resiliency, performance, time-based data retention	Limited scalability, implementation time

Next steps

Here are resources to help you design your storage solution and investigate its business aspects, including costs and service-level agreements.

Design storage solutions

- [Build great solutions with the Microsoft Azure Well-Architected Framework](#)
- [Understand data store models](#)
- [Select an Azure data store for your application](#)
- [Criteria for choosing a data store](#)
- [Choose a data storage approach in Azure](#)
- [Developing with Azure Cosmos DB Table API and Azure Table storage](#)

Azure service limits, cost, service level agreements (SLA), and regional availability

- [Azure subscription and service limits, quotas, and constraints](#)
- [Azure pricing](#)
- [Service-level agreements](#)
- [Products available by region](#)

Choose an analytical data store in Azure

3/10/2022 • 5 minutes to read • [Edit Online](#)

In a [big data](#) architecture, there is often a need for an analytical data store that serves processed data in a structured format that can be queried using analytical tools. Analytical data stores that support querying of both hot-path and cold-path data are collectively referred to as the serving layer, or data serving storage.

The serving layer deals with processed data from both the hot path and cold path. In the [lambda architecture](#), the serving layer is subdivided into a *speed serving* layer, which stores data that has been processed incrementally, and a *batch serving* layer, which contains the batch-processed output. The serving layer requires strong support for random reads with low latency. Data storage for the speed layer should also support random writes, because batch loading data into this store would introduce undesired delays. On the other hand, data storage for the batch layer does not need to support random writes, but batch writes instead.

There is no single best data management choice for all data storage tasks. Different data management solutions are optimized for different tasks. Most real-world cloud apps and big data processes have a variety of data storage requirements and often use a combination of data storage solutions.

What are your options when choosing an analytical data store?

There are several options for data serving storage in Azure, depending on your needs:

- [Azure Synapse Analytics](#)
- [Azure Synapse Spark pools](#)
- [Azure Databricks](#)
- [Azure Data Explorer](#)
- [Azure SQL Database](#)
- [SQL Server in Azure VM](#)
- [HBase/Phoenix on HDInsight](#)
- [Hive LLAP on HDInsight](#)
- [Azure Analysis Services](#)
- [Azure Cosmos DB](#)

These options provide various database models that are optimized for different types of tasks:

- [Key/value](#) databases hold a single serialized object for each key value. They're good for storing large volumes of data where you want to get one item for a given key value and you don't have to query based on other properties of the item.
- [Document](#) databases are key/value databases in which the values are *documents*. A "document" in this context is a collection of named fields and values. The database typically stores the data in a format such as XML, YAML, JSON, or BSON, but may use plain text. Document databases can query on non-key fields and define secondary indexes to make querying more efficient. This makes a document database more suitable for applications that need to retrieve data based on criteria more complex than the value of the document key. For example, you could query on fields such as product ID, customer ID, or customer name.
- [Column-family](#) databases are key/value data stores that structure data storage into collections of related columns called column families. For example, a census database might have one group of columns for a person's name (first, middle, last), one group for the person's address, and one group for the person's profile information (date of birth, gender). The database can store each column family in a separate partition, while keeping all of the data for one person related to the same key. An application can read a single column family

without reading through all of the data for an entity.

- **Graph** databases store information as a collection of objects and relationships. A graph database can efficiently perform queries that traverse the network of objects and the relationships between them. For example, the objects might be employees in a human resources database, and you might want to facilitate queries such as "find all employees who directly or indirectly work for Scott."
- Telemetry and time series databases are an append-only collection of objects. Telemetry databases efficiently index data in a variety of column stores and in-memory structures, making them the optimal choice for storing and analyzing vast quantities of telemetry and time series data.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need serving storage that can serve as a hot path for your data? If yes, narrow your options to those that are optimized for a speed serving layer.
- Do you need massively parallel processing (MPP) support, where queries are automatically distributed across several processes or nodes? If yes, select an option that supports query scale out.
- Do you prefer to use a relational data store? If so, narrow your options to those with a relational database model. However, note that some non-relational stores support SQL syntax for querying, and tools such as PolyBase can be used to query non-relational data stores.
- Do you collect time series data? Do you use append-only data?

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CapabilitY	SQL DatabasE	Azure Synapse SQL Pool	Azure Synapse Spark Pool	Azure Data Explorer	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Is managed service	Yes	Yes	Yes	Yes	Yes ¹	Yes ¹	Yes	Yes
Primary database model	Relational (columnar format when using columnstore indexes)	Relational tables with columnar storage	Wide column store	Relational (column store), telemetry, and time series store	Wide column store	Hive/In-Memory	Tabular semantic models	Document store, graph, key-value store, wide column store
SQL language support	Yes	Yes	Yes	Yes	Yes (using Phoenix JDBC driver)	Yes	No	Yes

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Optimized for speed serving layer	Yes ²	Yes ³	Yes	Yes	Yes	Yes	No	Yes

[1] With manual configuration and scaling.

[2] Using memory-optimized tables and hash or nonclustered indexes.

[3] Supported as an Azure Stream Analytics output.

Scalability capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE SQL POOL	AZURE SYNAPSE SPARK POOL	AZURE DATA EXPLORER	HBASE/P HOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Redundant regional servers for high availability	Yes	No	No	Yes	Yes	No	No	Yes
Supports query scale out	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic scalability (scale up)	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Supports in-memory caching of data	Yes	Yes	Yes	Yes	No	Yes	Yes	No

Security capabilities

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Authentication	SQL / Azure Active Directory (Azure AD)	SQL / Azure AD	Azure AD	local / Azure AD ¹	local / Azure AD ¹	Azure AD	database users / Azure AD via access control (IAM)

CAPABILITY	SQL DATABASE	AZURE SYNAPSE	AZURE DATA EXPLORER	HBASE/PHOENIX ON HDINSIGHT	HIVE LLAP ON HDINSIGHT	AZURE ANALYSIS SERVICES	COSMOS DB
Data encryption at rest	Yes ²	Yes ²	Yes	Yes ¹	Yes ¹	Yes	Yes
Row-level security	Yes	Yes ³	No	Yes ¹	Yes ¹	Yes	No
Supports firewalls	Yes	Yes	Yes	Yes ⁴	Yes ⁴	Yes	Yes
Dynamic data masking	Yes	Yes	Yes	Yes ¹	Yes	No	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Requires using transparent data encryption (TDE) to encrypt and decrypt your data at rest.

[3] Filter predicates only. See [Row-Level Security](#)

[4] When used within an Azure Virtual Network. See [Extend Azure HDInsight using an Azure Virtual Network](#).

Choose a data analytics technology in Azure

3/10/2022 • 4 minutes to read • [Edit Online](#)

The goal of most big data solutions is to provide insights into the data through analysis and reporting. This can include preconfigured reports and visualizations, or interactive data exploration.

What are your options when choosing a data analytics technology?

There are several options for analysis, visualizations, and reporting in Azure, depending on your needs:

- [Power BI](#)
- [Jupyter Notebooks](#)
- [Zeppelin Notebooks](#)
- [Microsoft Azure Notebooks](#)

Power BI

[Power BI](#) is a suite of business analytics tools. It can connect to hundreds of data sources, and can be used for ad hoc analysis. See [this list](#) of the currently available data sources. Use [Power BI Embedded](#) to integrate Power BI within your own applications without requiring any additional licensing.

Organizations can use Power BI to produce reports and publish them to the organization. Everyone can create personalized dashboards, with governance and [security built in](#). Power BI uses [Azure Active Directory](#) (Azure AD) to authenticate users who log in to the Power BI service, and uses the Power BI login credentials whenever a user attempts to access resources that require authentication.

Jupyter Notebooks

[Jupyter Notebooks](#) provide a browser-based shell that lets data scientists create *notebook* files that contain Python, Scala, or R code and markdown text, making it an effective way to collaborate by sharing and documenting code and results in a single document.

Most varieties of HDInsight clusters, such as Spark or Hadoop, come [preconfigured with Jupyter notebooks](#) for interacting with data and submitting jobs for processing. Depending on the type of HDInsight cluster you are using, one or more kernels will be provided for interpreting and running your code. For example, Spark clusters on HDInsight provide Spark-related kernels that you can select from to execute Python or Scala code using the Spark engine.

Jupyter notebooks provide a great environment for analyzing, visualizing, and processing your data prior to building more advanced visualizations with a BI/reporting tool like Power BI.

Zeppelin Notebooks

[Zeppelin Notebooks](#) are another option for a browser-based shell, similar to Jupyter in functionality. Some HDInsight clusters come [preconfigured with Zeppelin notebooks](#). However, if you are using an [HDInsight Interactive Query](#) (Hive LLAP) cluster, [Zeppelin](#) is currently your only choice of notebook that you can use to run interactive Hive queries. Also, if you are using a [domain-joined HDInsight cluster](#), Zeppelin notebooks are the only type that enables you to assign different user logins to control access to notebooks and the underlying Hive tables.

Microsoft Azure Notebooks

[Azure Notebooks](#) is an online Jupyter Notebooks-based service that enables data scientists to create, run, and share Jupyter Notebooks in cloud-based libraries. Azure Notebooks provides execution environments for Python 2, Python 3, F#, and R, and provides several charting libraries for visualizing your data, such as ggplot,

matplotlib, bokeh, and seaborn.

Unlike Jupyter notebooks running on an HDInsight cluster, which are connected to the cluster's default storage account, Azure Notebooks does not provide any data. You must [load data](#) in a variety of ways, such downloading data from an online source, interacting with Azure Blobs or Table Storage, connecting to a SQL database, or loading data with the Copy Wizard for Azure Data Factory.

Key benefits:

- Free service—no Azure subscription required.
- No need to install Jupyter and the supporting R or Python distributions locally—just use a browser.
- Manage your own online libraries and access them from any device.
- Share your notebooks with collaborators.

Considerations:

- You will be unable to access your notebooks when offline.
- Limited processing capabilities of the free notebook service may not be enough to train large or complex models.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need to connect to numerous data sources, providing a centralized place to create reports for data spread throughout your domain? If so, choose an option that allows you to connect to 100s of data sources.
- Do you want to embed dynamic visualizations in an external website or application? If so, choose an option that provides embedding capabilities.
- Do you want to design your visualizations and reports while offline? If yes, choose an option with offline capabilities.
- Do you need heavy processing power to train large or complex AI models or work with very large data sets? If yes, choose an option that can connect to a big data cluster.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Connect to big data cluster for advanced processing	Yes	Yes	Yes	No
Managed service	Yes	Yes ¹	Yes ¹	Yes
Connect to 100s of data sources	Yes	No	No	No
Offline capabilities	Yes ²	No	No	No

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Embedding capabilities	Yes	No	No	No
Automatic data refresh	Yes	No	No	No
Access to numerous open source packages	No	Yes ³	Yes ³	Yes ⁴
Data transformation/cleaning options	Power Query, R	40 languages, including Python, R, Julia, and Scala	20+ interpreters, including Python, JDBC, and R	Python, F#, R
Pricing	Free for Power BI Desktop (authoring), see pricing for hosting options	Free	Free	Free
Multiuser collaboration	Yes	Yes (through sharing or with a multiuser server like JupyterHub)	Yes	Yes (through sharing)

[1] When used as part of a managed HDInsight cluster.

[2] With the use of Power BI Desktop.

[2] You can search the [Maven repository](#) for community-contributed packages.

[3] Python packages can be installed using either pip or conda. R packages can be installed from CRAN or GitHub. Packages in F# can be installed via nuget.org using the [Paket dependency manager](#).

Choose a batch processing technology in Azure

3/10/2022 • 3 minutes to read • [Edit Online](#)

Big data solutions often use long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files from scalable storage (like HDFS, Azure Data Lake Store, and Azure Storage), processing them, and writing the output to new files in scalable storage.

The key requirement of such batch processing engines is the ability to scale out computations, in order to handle a large volume of data. Unlike real-time processing, however, batch processing is expected to have latencies (the time between data ingestion and computing a result) that measure in minutes to hours.

Technology choices for batch processing

Azure Synapse Analytics

[Azure Synapse](#) is a distributed system designed to perform analytics on large data. It supports massive parallel processing (MPP), which makes it suitable for running high-performance analytics. Consider Azure Synapse when you have large amounts of data (more than 1 TB) and are running an analytics workload that will benefit from parallelism.

Azure Data Lake Analytics

[Data Lake Analytics](#) is an on-demand analytics job service. It is optimized for distributed processing of very large data sets stored in Azure Data Lake Store.

- Languages: [U-SQL](#) (including Python, R, and C# extensions).
- Integrates with Azure Data Lake Store, Azure Storage blobs, Azure SQL Database, and Azure Synapse.
- Pricing model is per-job.

HDInsight

HDInsight is a managed Hadoop service. Use it to deploy and manage Hadoop clusters in Azure. For batch processing, you can use [Spark](#), [Hive](#), [Hive LLAP](#), [MapReduce](#).

- Languages: R, Python, Java, Scala, SQL
- Kerberos authentication with Active Directory, Apache Ranger based access control
- Gives you full control of the Hadoop cluster

Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform. You can think of it as "Spark as a service." It's the easiest way to use Spark on the Azure platform.

- Languages: R, Python, Java, Scala, Spark SQL
- Fast cluster start times, autotermination, autoscaling.
- Manages the Spark cluster for you.
- Built-in integration with Azure Blob Storage, Azure Data Lake Storage (ADLS), Azure Synapse, and other services. See [Data Sources](#).
- User authentication with Azure Active Directory.
- Web-based [notebooks](#) for collaboration and data exploration.
- Supports [GPU-enabled clusters](#)

Azure Distributed Data Engineering Toolkit

The [Distributed Data Engineering Toolkit](#) (AZTK) is a tool for provisioning on-demand Spark on Docker clusters

in Azure.

AZTK is not an Azure service. Rather, it's a client-side tool with a CLI and Python SDK interface, that's built on Azure Batch. This option gives you the most control over the infrastructure when deploying a Spark cluster.

- Bring your own Docker image.
- Use low-priority VMs for an 80% discount.
- Mixed mode clusters that use both low-priority and dedicated VMs.
- Built in support for Azure Blob Storage and Azure Data Lake connection.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Do you want to author batch processing logic declaratively or imperatively?
- Will you perform batch processing in bursts? If yes, consider options that let you auto-terminate the cluster or whose pricing model is per batch job.
- Do you need to query relational data stores along with your batch processing, for example to look up reference data? If yes, consider the options that enable querying of external relational stores.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT	AZURE DATABRICKS
Is managed service	Yes	Yes	Yes ¹	Yes
Relational data store	Yes	Yes	No	No
Pricing model	Per batch job	By cluster hour	By cluster hour	Databricks Unit ² + cluster hour

[1] With manual configuration.

[2] A Databricks Unit (DBU) is a unit of processing capability per hour.

Capabilities

CAPABILITY	AZURE DATA LAKE ANALYTICS	AZURE SYNAPSE	HDINSIGHT WITH SPARK	HDINSIGHT WITH HIVE	HDINSIGHT WITH HIVE LLAP	AZURE DATABRICKS
Autoscaling	No	No	Yes	Yes	Yes	Yes
Scale-out granularity	Per job	Per cluster	Per cluster	Per cluster	Per cluster	Per cluster
In-memory caching of data	No	Yes	Yes	No	Yes	Yes

Capability	Azure Data Lake Analytics	Azure Synapse	HDIInsight with Spark	HDIInsight with Hive	HDIInsight with Hive LLAP	Azure Databricks
Query from external relational stores	Yes	No	Yes	No	No	Yes
Authentication	Azure AD	SQL / Azure AD	No	Azure AD ¹	Azure AD ¹	Azure AD
Auditing	Yes	Yes	No	Yes ¹	Yes ¹	Yes
Row-level security	No	Yes ²	No	Yes ¹	Yes ¹	No
Supports firewalls	Yes	Yes	Yes	Yes ³	Yes ³	No
Dynamic data masking	No	Yes	No	Yes ¹	Yes ¹	No

[1] Requires using a [domain-joined HDInsight cluster](#).

[2] Filter predicates only. See [Row-Level Security](#)

[3] Supported when [used within an Azure Virtual Network](#).

Next steps

- [Analytics architecture design](#)
- [Choose an analytical data store in Azure](#)
- [Choose a data analytics technology in Azure](#)
- [Analytics end-to-end with Azure Synapse](#)

Data lakes

3/10/2022 • 2 minutes to read • [Edit Online](#)

A data lake is a storage repository that holds a large amount of data in its native, raw format. Data lake stores are optimized for scaling to terabytes and petabytes of data. The data typically comes from multiple heterogeneous sources, and may be structured, semi-structured, or unstructured. The idea with a data lake is to store everything in its original, untransformed state. This approach differs from a traditional [data warehouse](#), which transforms and processes the data at the time of ingestion.

Advantages of a data lake:

- Data is never thrown away, because the data is stored in its raw format. This is especially useful in a big data environment, when you may not know in advance what insights are available from the data.
- Users can explore the data and create their own queries.
- May be faster than traditional ETL tools.
- More flexible than a data warehouse, because it can store unstructured and semi-structured data.

A complete data lake solution consists of both storage and processing. Data lake storage is designed for fault-tolerance, infinite scalability, and high-throughput ingestion of data with varying shapes and sizes. Data lake processing involves one or more processing engines built with these goals in mind, and can operate on data stored in a data lake at scale.

When to use a data lake

Typical uses for a data lake include [data exploration](#), data analytics, and machine learning.

A data lake can also act as the data source for a data warehouse. With this approach, the raw data is ingested into the data lake and then transformed into a structured queryable format. Typically this transformation uses an [ELT](#) (extract-load-transform) pipeline, where the data is ingested and transformed in place. Source data that is already relational may go directly into the data warehouse, using an ETL process, skipping the data lake.

Data lake stores are often used in event streaming or IoT scenarios, because they can persist large amounts of relational and nonrelational data without transformation or schema definition. They are built to handle high volumes of small writes at low latency, and are optimized for massive throughput.

Challenges

- Lack of a schema or descriptive metadata can make the data hard to consume or query.
- Lack of semantic consistency across the data can make it challenging to perform analysis on the data, unless users are highly skilled at data analytics.
- It can be hard to guarantee the quality of the data going into the data lake.
- Without proper governance, access control and privacy issues can be problems. What information is going into the data lake, who can access that data, and for what uses?
- A data lake may not be the best way to integrate data that is already relational.
- By itself, a data lake does not provide integrated or holistic views across the organization.
- A data lake may become a dumping ground for data that is never actually analyzed or mined for insights.

Relevant Azure services

- [Data Lake Store](#) is a hyperscale, Hadoop-compatible repository.

- [Data Lake Analytics](#) is an on-demand analytics job service to simplify big data analytics.

Choose a big data storage technology in Azure

3/10/2022 • 8 minutes to read • [Edit Online](#)

This topic compares options for data storage for big data solutions — specifically, data storage for bulk data ingestion and batch processing, as opposed to [analytical data stores](#) or [real-time streaming ingestion](#).

What are your options when choosing data storage in Azure?

There are several options for ingesting data into Azure, depending on your needs.

File storage:

- [Azure Storage blobs](#)
- [Azure Data Lake Store](#)

NoSQL databases:

- [Azure Cosmos DB](#)
- [HBase on HDInsight](#)

Analytical databases:

[Azure Data Explorer](#)

Azure Storage blobs

Azure Storage is a managed storage service that is highly available, secure, durable, scalable, and redundant. Microsoft takes care of maintenance and handles critical problems for you. Azure Storage is the most ubiquitous storage solution Azure provides, due to the number of services and tools that can be used with it.

There are various Azure Storage services you can use to store data. The most flexible option for storing blobs from a number of data sources is [Blob storage](#). Blobs are basically files. They store pictures, documents, HTML files, virtual hard disks (VHDs), big data such as logs, database backups — pretty much anything. Blobs are stored in containers, which are similar to folders. A container provides a grouping of a set of blobs. A storage account can contain an unlimited number of containers, and a container can store an unlimited number of blobs.

Azure Storage is a good choice for big data and analytics solutions, because of its flexibility, high availability, and low cost. It provides hot, cool, and archive storage tiers for different use cases. For more information, see [Azure Blob Storage: Hot, cool, and archive storage tiers](#).

Azure Blob storage can be accessed from Hadoop (available through HDInsight). HDInsight can use a blob container in Azure Storage as the default file system for the cluster. Through a Hadoop distributed file system (HDFS) interface provided by a WASB driver, the full set of components in HDInsight can operate directly on structured or unstructured data stored as blobs. Azure Blob storage can also be accessed via Azure Synapse Analytics using its PolyBase feature.

Other features that make Azure Storage a good choice are:

- [Multiple concurrency strategies](#).
- [Disaster recovery and high availability options](#).
- [Encryption at rest](#).
- [Azure role-based access control \(Azure RBAC\)](#) to control access using Azure Active Directory users and groups.

Azure Data Lake Store

[Azure Data Lake Store](#) is an enterprise-wide hyperscale repository for big data analytic workloads. Data Lake enables you to capture data of any size, type, and ingestion speed in one single [secure](#) location for operational and exploratory analytics.

Data Lake Store does not impose any limits on account sizes, file sizes, or the amount of data that can be stored in a data lake. Data is stored durably by making multiple copies and there is no limit on the duration of time that the data can be stored in the Data Lake. In addition to making multiple copies of files to guard against any unexpected failures, Data lake spreads parts of a file over a number of individual storage servers. This improves the read throughput when reading the file in parallel for performing data analytics.

Data Lake Store can be accessed from Hadoop (available through HDInsight) using the WebHDFS-compatible REST APIs. You may consider using this as an alternative to Azure Storage when your individual or combined file sizes exceed that which is supported by Azure Storage. However, there are [performance tuning guidelines](#) you should follow when using Data Lake Store as your primary storage for an HDInsight cluster, with specific guidelines for [Spark](#), [Hive](#), [MapReduce](#), and [Storm](#). Also, be sure to check Data Lake Store's [regional availability](#), because it is not available in as many regions as Azure Storage, and it needs to be located in the same region as your HDInsight cluster.

Coupled with Azure Data Lake Analytics, Data Lake Store is specifically designed to enable analytics on the stored data and is tuned for performance for data analytics scenarios. Data Lake Store can also be accessed via Azure Synapse using its PolyBase feature.

Azure Cosmos DB

[Azure Cosmos DB](#) is Microsoft's globally distributed multi-model database. Cosmos DB guarantees single-digit-millisecond latencies at the 99th percentile anywhere in the world, offers multiple well-defined consistency models to fine-tune performance, and guarantees high availability with multi-homing capabilities.

Azure Cosmos DB is schema-agnostic. It automatically indexes all the data without requiring you to deal with schema and index management. It's also multi-model, natively supporting document, key-value, graph, and column-family data models.

Azure Cosmos DB features:

- [Geo-replication](#)
- [Elastic scaling of throughput and storage](#) worldwide
- [Five well-defined consistency levels](#)

HBase on HDInsight

[Apache HBase](#) is an open-source, NoSQL database that is built on Hadoop and modeled after Google BigTable. HBase provides random access and strong consistency for large amounts of unstructured and semi-structured data in a schemaless database organized by column families.

Data is stored in the rows of a table, and data within a row is grouped by column family. HBase is schemaless in the sense that neither the columns nor the type of data stored in them need to be defined before using them. The open-source code scales linearly to handle petabytes of data on thousands of nodes. It can rely on data redundancy, batch processing, and other features that are provided by distributed applications in the Hadoop ecosystem.

The [HDInsight implementation](#) leverages the scale-out architecture of HBase to provide automatic sharding of tables, strong consistency for reads and writes, and automatic failover. Performance is enhanced by in-memory caching for reads and high-throughput streaming for writes. In most cases, you'll want to [create the HBase cluster inside a virtual network](#) so other HDInsight clusters and applications can directly access the tables.

Azure Data Explorer

[Azure Data Explorer](#) is a fast and highly scalable data exploration service for log and telemetry data. It helps you handle the many data streams emitted by modern software so you can collect, store, and analyze data. Azure Data Explorer is ideal for analyzing large volumes of diverse data from any data source, such as websites, applications, IoT devices, and more. This data is used for diagnostics, monitoring, reporting, machine learning, and additional analytics capabilities. Azure Data Explorer makes it simple to ingest this data and enables you to do complex ad hoc queries on the data in seconds.

Azure Data Explorer can be linearly [scaled out](#) for increasing ingestion and query processing throughput. An Azure Data Explorer cluster can be [deployed to a Virtual Network](#) for enabling private networks.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need managed, high-speed, cloud-based storage for any type of text or binary data? If yes, then select one of the file storage or analytics options.
- Do you need file storage that is optimized for parallel analytics workloads and high throughput/IOPS? If yes, then choose an option that is tuned to analytics workload performance.
- Do you need to store unstructured or semi-structured data in a schemaless database? If so, select one of the non-relational or analytics options. Compare options for indexing and database models. Depending on the type of data you need to store, the primary database models may be the largest factor.
- Can you use the service in your region? Check the regional availability for each Azure service. See [Products available by region](#).

Capability matrix

The following tables summarize the key differences in capabilities.

File storage capabilities

CAPABILITY	AZURE DATA LAKE STORE	AZURE BLOB STORAGE CONTAINERS
Purpose	Optimized storage for big data analytics workloads	General purpose object store for a wide variety of storage scenarios
Use cases	Batch, streaming analytics, and machine learning data such as log files, IoT data, click streams, large datasets	Any type of text or binary data, such as application back end, backup data, media storage for streaming, and general purpose data
Structure	Hierarchical file system	Object store with flat namespace
Authentication	Based on Azure Active Directory Identities	Based on shared secrets Account Access Keys and Shared Access Signature Keys , and Azure role-based access control (Azure RBAC)
Authentication protocol	OAuth 2.0. Calls must contain a valid JWT (JSON web token) issued by Azure Active Directory	Hash-based message authentication code (HMAC). Calls must contain a Base64-encoded SHA-256 hash over a part of the HTTP request.

Capability	Azure Data Lake Store	Azure Blob Storage Containers
Authorization	POSIX access control lists (ACLs). ACLs based on Azure Active Directory identities can be set file and folder level.	For account-level authorization use Account Access Keys . For account, container, or blob authorization use Shared Access Signature Keys .
Auditing	Available.	Available
Encryption at rest	Transparent, server side	Transparent, server side; Client-side encryption
Developer SDKs	.NET, Java, Python, Node.js	.NET, Java, Python, Node.js, C++, Ruby
Analytics workload performance	Optimized performance for parallel analytics workloads, High Throughput and IOPS	Not optimized for analytics workloads
Size limits	No limits on account sizes, file sizes or number of files	Specific limits documented here
Geo-redundancy	Locally-redundant (LRS), globally redundant (GRS), read-access globally redundant (RA-GRS), zone-redundant (ZRS).	Locally redundant (LRS), globally redundant (GRS), read-access globally redundant (RA-GRS), zone-redundant (ZRS). See here for more information

NoSQL database capabilities

Capability	Azure Cosmos DB	HBase on HDInsight
Primary database model	Document store, graph, key-value store, wide column store	Wide column store
Secondary indexes	Yes	No
SQL language support	Yes	Yes (using the Phoenix JDBC driver)
Consistency	Strong, bounded-staleness, session, consistent prefix, eventual	Strong
Native Azure Functions integration	Yes	No
Automatic global distribution	Yes	No HBase cluster replication can be configured across regions with eventual consistency
Pricing model	Elastically scalable request units (RUs) charged per-second as needed, elastically scalable storage	Per-minute pricing for HDInsight cluster (horizontal scaling of nodes), storage

Analytical database capabilities

Capability	Azure Data Explorer

CAPABILITY	AZURE DATA EXPLORER
Primary database model	Relational (column store), telemetry, and time series store
SQL language support	Yes
Pricing model	Elastically scalable cluster instances
Authentication	Based on Azure Active Directory identities
Encryption at rest	Supported, customer managed keys
Analytics workload performance	Optimized performance for parallel analytics workloads
Size limits	Linearly scalable

Understand data store models

3/10/2022 • 12 minutes to read • [Edit Online](#)

Modern business systems manage increasingly large volumes of heterogeneous data. This heterogeneity means that a single data store is usually not the best approach. Instead, it's often better to store different types of data in different data stores, each focused toward a specific workload or usage pattern. The term *polyglot persistence* is used to describe solutions that use a mix of data store technologies. Therefore, it's important to understand the main storage models and their tradeoffs.

Selecting the right data store for your requirements is a key design decision. There are literally hundreds of implementations to choose from among SQL and NoSQL databases. Data stores are often categorized by how they structure data and the types of operations they support. This article describes several of the most common storage models. Note that a particular data store technology may support multiple storage models. For example, a relational database management systems (RDBMS) may also support key/value or graph storage. In fact, there is a general trend for so-called *multi-model* support, where a single database system supports several models. But it's still useful to understand the different models at a high level.

Not all data stores in a given category provide the same feature-set. Most data stores provide server-side functionality to query and process data. Sometimes this functionality is built into the data storage engine. In other cases, the data storage and processing capabilities are separated, and there may be several options for processing and analysis. Data stores also support different programmatic and management interfaces.

Generally, you should start by considering which storage model is best suited for your requirements. Then consider a particular data store within that category, based on factors such as feature set, cost, and ease of management.

NOTE

Learn more about identifying and reviewing your data service requirements for cloud adoption, in the [Microsoft Cloud Adoption Framework for Azure](#). Likewise, you can also learn about [selecting storage tools and services](#).

Relational database management systems

Relational databases organize data as a series of two-dimensional tables with rows and columns. Most vendors provide a dialect of the Structured Query Language (SQL) for retrieving and managing data. An RDBMS typically implements a transactionally consistent mechanism that conforms to the ACID (Atomic, Consistent, Isolated, Durable) model for updating information.

An RDBMS typically supports a schema-on-write model, where the data structure is defined ahead of time, and all read or write operations must use the schema.

This model is very useful when strong consistency guarantees are important — where all changes are atomic, and transactions always leave the data in a consistent state. However, an RDBMS generally can't scale out horizontally without sharding the data in some way. Also, the data in an RDBMS must be normalized, which isn't appropriate for every data set.

Azure services

- [Azure SQL Database | \(Security Baseline\)](#)
- [Azure Database for MySQL | \(Security Baseline\)](#)
- [Azure Database for PostgreSQL | \(Security Baseline\)](#)

- [Azure Database for MariaDB | \(Security Baseline\)](#)

Workload

- Records are frequently created and updated.
- Multiple operations have to be completed in a single transaction.
- Relationships are enforced using database constraints.
- Indexes are used to optimize query performance.

Data type

- Data is highly normalized.
- Database schemas are required and enforced.
- Many-to-many relationships between data entities in the database.
- Constraints are defined in the schema and imposed on any data in the database.
- Data requires high integrity. Indexes and relationships need to be maintained accurately.
- Data requires strong consistency. Transactions operate in a way that ensures all data are 100% consistent for all users and processes.
- Size of individual data entries is small to medium-sized.

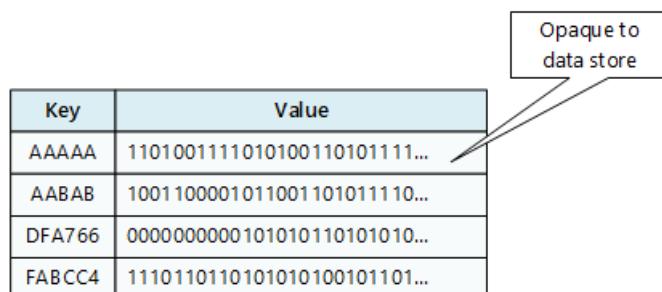
Examples

- Inventory management
- Order management
- Reporting database
- Accounting

Key/value stores

A key/value store associates each data value with a unique key. Most key/value stores only support simple query, insert, and delete operations. To modify a value (either partially or completely), an application must overwrite the existing data for the entire value. In most implementations, reading or writing a single value is an atomic operation.

An application can store arbitrary data as a set of values. Any schema information must be provided by the application. The key/value store simply retrieves or stores the value by key.



Key	Value
AAAAAA	11010011101010011010111...
AABAB	100110000101100110101110...
DFA766	00000000001010101101010...
FABCC4	11101101101010100101101...

Key/value stores are highly optimized for applications performing simple lookups, but are less suitable if you need to query data across different key/value stores. Key/value stores are also not optimized for querying by value.

A single key/value store can be extremely scalable, as the data store can easily distribute data across multiple nodes on separate machines.

Azure services

- [Azure Cosmos DB Table API and SQL API | \(Cosmos DB Security Baseline\)](#)
- [Azure Cache for Redis | \(Security Baseline\)](#)
- [Azure Table Storage | \(Security Baseline\)](#)

Workload

- Data is accessed using a single key, like a dictionary.
- No joins, lock, or unions are required.
- No aggregation mechanisms are used.
- Secondary indexes are generally not used.

Data type

- Each key is associated with a single value.
- There is no schema enforcement.
- No relationships between entities.

Examples

- Data caching
- Session management
- User preference and profile management
- Product recommendation and ad serving

Document databases

A document database stores a collection of *documents*, where each document consists of named fields and data. The data can be simple values or complex elements such as lists and child collections. Documents are retrieved by unique keys.

Typically, a document contains the data for single entity, such as a customer or an order. A document may contain information that would be spread across several relational tables in an RDBMS. Documents don't need to have the same structure. Applications can store different data in documents as business requirements change.

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Azure service

- [Azure Cosmos DB SQL API | \(Cosmos DB Security Baseline\)](#)

Workload

- Insert and update operations are common.
- No object-relational impedance mismatch. Documents can better match the object structures used in application code.
- Individual documents are retrieved and written as a single block.

- Data requires index on multiple fields.

Data type

- Data can be managed in de-normalized way.
- Size of individual document data is relatively small.
- Each document type can use its own schema.
- Documents can include optional fields.
- Document data is semi-structured, meaning that data types of each field are not strictly defined.

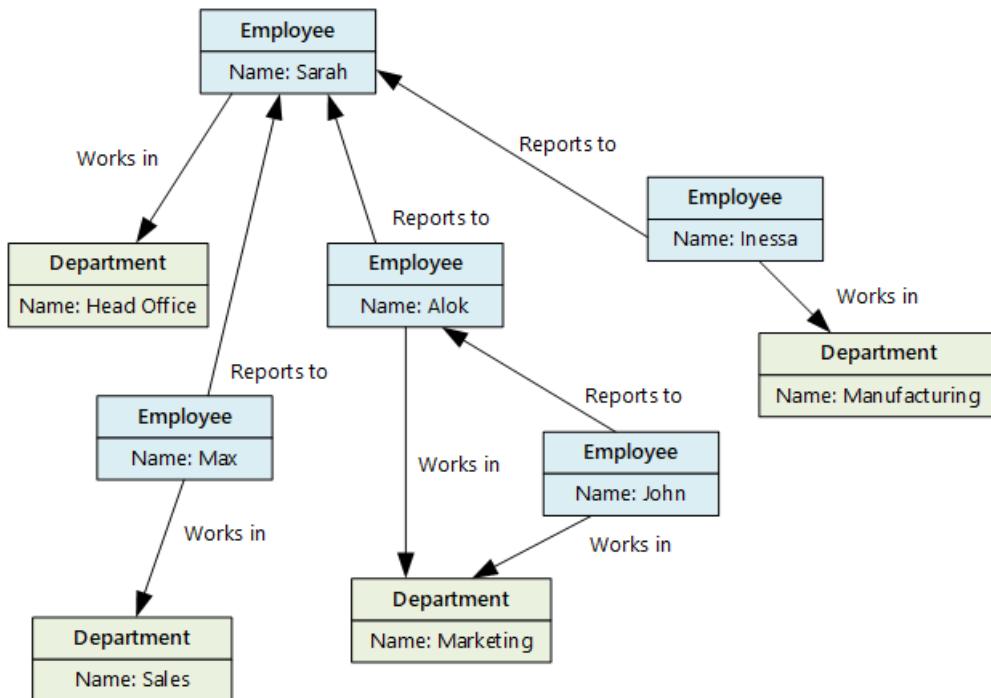
Examples

- Product catalog
- Content management
- Inventory management

Graph databases

A graph database stores two types of information, nodes and edges. Edges specify relationships between nodes. Nodes and edges can have properties that provide information about that node or edge, similar to columns in a table. Edges can also have a direction indicating the nature of the relationship.

Graph databases can efficiently perform queries across the network of nodes and edges and analyze the relationships between entities. The following diagram shows an organization's personnel database structured as a graph. The entities are employees and departments, and the edges indicate reporting relationships and the departments in which employees work.



This structure makes it straightforward to perform queries such as "Find all employees who report directly or indirectly to Sarah" or "Who works in the same department as John?" For large graphs with lots of entities and relationships, you can perform very complex analyses very quickly. Many graph databases provide a query language that you can use to traverse a network of relationships efficiently.

Azure services

- [Azure Cosmos DB Gremlin API | \(Security Baseline\)](#)
- [SQL Server | \(Security Baseline\)](#)

Workload

- Complex relationships between data items involving many hops between related data items.
- The relationship between data items are dynamic and change over time.
- Relationships between objects are first-class citizens, without requiring foreign-keys and joins to traverse.

Data type

- Nodes and relationships.
- Nodes are similar to table rows or JSON documents.
- Relationships are just as important as nodes, and are exposed directly in the query language.
- Composite objects, such as a person with multiple phone numbers, tend to be broken into separate, smaller nodes, combined with traversable relationships

Examples

- Organization charts
- Social graphs
- Fraud detection
- Recommendation engines

Data analytics

Data analytics stores provide massively parallel solutions for ingesting, storing, and analyzing data. The data is distributed across multiple servers to maximize scalability. Large data file formats such as delimiter files (CSV), [parquet](#), and [ORC](#) are widely used in data analytics. Historical data is typically stored in data stores such as blob storage or [Azure Data Lake Storage Gen2](#), which are then accessed by Azure Synapse, Databricks, or HDInsight as external tables. A typical scenario using data stored as parquet files for performance, is described in the article [Use external tables with Synapse SQL](#).

Azure services

- [Azure Synapse Analytics | \(Security Baseline\)](#)
- [Azure Data Lake | \(Security Baseline\)](#)
- [Azure Data Explorer | \(Security Baseline\)](#)
- [Azure Analysis Services](#)
- [HDInsight | \(Security Baseline\)](#)
- [Azure Databricks | \(Security Baseline\)](#)

Workload

- Data analytics
- Enterprise BI

Data type

- Historical data from multiple sources.
- Usually denormalized in a "star" or "snowflake" schema, consisting of fact and dimension tables.
- Usually loaded with new data on a scheduled basis.
- Dimension tables often include multiple historic versions of an entity, referred to as a *slowly changing dimension*.

Examples

- Enterprise data warehouse

Column-family databases

A column-family database organizes data into rows and columns. In its simplest form, a column-family database can appear very similar to a relational database, at least conceptually. The real power of a column-family

database lies in its denormalized approach to structuring sparse data.

You can think of a column-family database as holding tabular data with rows and columns, but the columns are divided into groups known as *column families*. Each column family holds a set of columns that are logically related together and are typically retrieved or manipulated as a unit. Other data that is accessed separately can be stored in separate column families. Within a column family, new columns can be added dynamically, and rows can be sparse (that is, a row doesn't need to have a value for every column).

The following diagram shows an example with two column families, `Identity` and `Contact Info`. The data for a single entity has the same row key in each column-family. This structure, where the rows for any given object in a column family can vary dynamically, is an important benefit of the column-family approach, making this form of data store highly suited for storing structured, volatile data.

CustomerID	Column Family: Identity	CustomerID	Column Family: Contact Info
001	First name: Mu Bae Last name: Min	001	Phone number: 555-0100 Email: someone@example.com
002	First name: Francisco Last name: Vila Nova Suffix: Jr.	002	Email: vilanova@contoso.com
003	First name: Lena Last name: Adamczyz Title: Dr.	003	Phone number: 555-0120

Unlike a key/value store or a document database, most column-family databases store data in key order, rather than by computing a hash. Many implementations allow you to create indexes over specific columns in a column-family. Indexes let you retrieve data by columns value, rather than row key.

Read and write operations for a row are usually atomic with a single column-family, although some implementations provide atomicity across the entire row, spanning multiple column-families.

Azure services

- [Azure Cosmos DB Cassandra API | \(Security Baseline\)](#)
- [HBase in HDInsight | \(Security Baseline\)](#)

Workload

- Most column-family databases perform write operations extremely quickly.
- Update and delete operations are rare.
- Designed to provide high throughput and low-latency access.
- Supports easy query access to a particular set of fields within a much larger record.
- Massively scalable.

Data type

- Data is stored in tables consisting of a key column and one or more column families.
- Specific columns can vary by individual rows.
- Individual cells are accessed via get and put commands
- Multiple rows are returned using a scan command.

Examples

- Recommendations
- Personalization
- Sensor data
- Telemetry
- Messaging
- Social media analytics

- Web analytics
- Activity monitoring
- Weather and other time-series data

Search Engine Databases

A search engine database allows applications to search for information held in external data stores. A search engine database can index massive volumes of data and provide near real-time access to these indexes.

Indexes can be multi-dimensional and may support free-text searches across large volumes of text data.

Indexing can be performed using a pull model, triggered by the search engine database, or using a push model, initiated by external application code.

Searching can be exact or fuzzy. A fuzzy search finds documents that match a set of terms and calculates how closely they match. Some search engines also support linguistic analysis that can return matches based on synonyms, genre expansions (for example, matching `dogs` to `pets`), and stemming (matching words with the same root).

Azure service

- [Azure Search | \(Security Baseline\)](#)

Workload

- Data indexes from multiple sources and services.
- Queries are ad-hoc and can be complex.
- Full text search is required.
- Ad hoc self-service query is required.

Data type

- Semi-structured or unstructured text
- Text with reference to structured data

Examples

- Product catalogs
- Site search
- Logging

Time series databases

Time series data is a set of values organized by time. Time series databases typically collect large amounts of data in real time from a large number of sources. Updates are rare, and deletes are often done as bulk operations. Although the records written to a time-series database are generally small, there are often a large number of records, and total data size can grow rapidly.

Azure service

- [Azure Time Series Insights](#)

Workload

- Records are generally appended sequentially in time order.
- An overwhelming proportion of operations (95-99%) are writes.
- Updates are rare.
- Deletes occur in bulk, and are made to contiguous blocks or records.
- Data is read sequentially in either ascending or descending time order, often in parallel.

Data type

- A timestamp is used as the primary key and sorting mechanism.
- Tags may define additional information about the type, origin, and other information about the entry.

Examples

- Monitoring and event telemetry.
- Sensor or other IoT data.

Object storage

Object storage is optimized for storing and retrieving large binary objects (images, files, video and audio streams, large application data objects and documents, virtual machine disk images). Large data files are also popularly used in this model, for example, delimiter file (CSV), [parquet](#), and [ORC](#). Object stores can manage extremely large amounts of unstructured data.

Azure service

- [Azure Blob Storage | \(Security Baseline\)](#)
- [Azure Data Lake Storage Gen2 | \(Security Baseline\)](#)

Workload

- Identified by key.
- Content is typically an asset such as a delimiter, image, or video file.
- Content must be durable and external to any application tier.

Data type

- Data size is large.
- Value is opaque.

Examples

- Images, videos, office documents, PDFs
- Static HTML, JSON, CSS
- Log and audit files
- Database backups

Shared files

Sometimes, using simple flat files can be the most effective means of storing and retrieving information. Using file shares enables files to be accessed across a network. Given appropriate security and concurrent access control mechanisms, sharing data in this way can enable distributed services to provide highly scalable data access for performing basic, low-level operations such as simple read and write requests.

Azure service

- [Azure Files | \(Security Baseline\)](#)

Workload

- Migration from existing apps that interact with the file system.
- Requires SMB interface.

Data type

- Files in a hierarchical set of folders.
- Accessible with standard I/O libraries.

Examples

- Legacy files

- Shared content accessible among a number of VMs or app instances

Aided with this understanding of different data storage models, the next step is to evaluate your workload and application, and decide which data store will meet your specific needs. Use the [data storage decision tree](#) to help with this process.

Select an Azure data store for your application

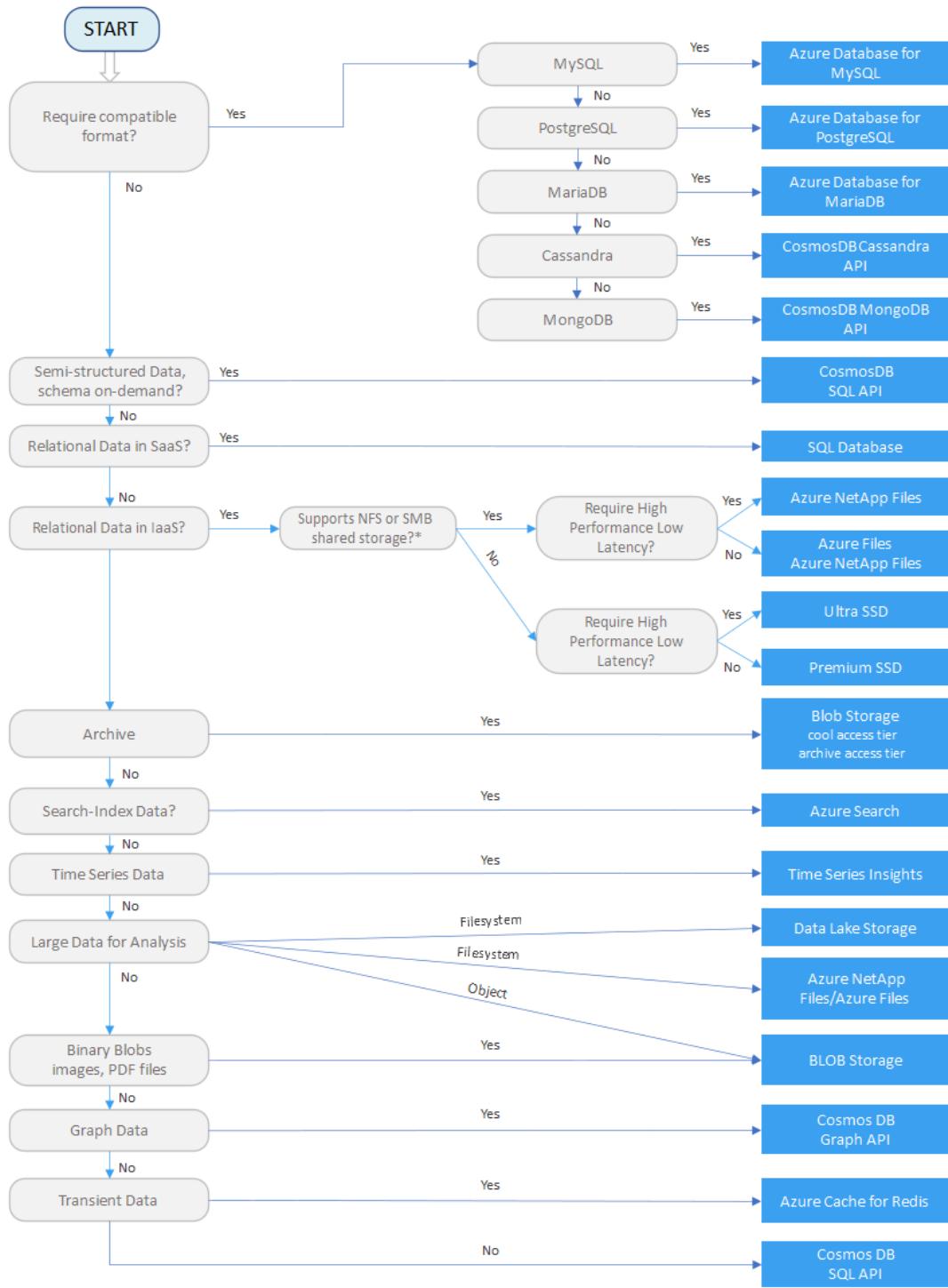
3/10/2022 • 2 minutes to read • [Edit Online](#)

Azure offers a number of managed data storage solutions, each providing different features and capabilities. This article will help you to choose a managed data store for your application.

If your application consists of multiple workloads, evaluate each workload separately. A complete solution may incorporate multiple data stores.

Select a candidate

Use the following flowchart to select a candidate Azure managed data store.



The output from this flowchart is a **starting point** for consideration. Next, perform a more detailed evaluation of the data store to see if it meets your needs. Refer to [Criteria for choosing a data store](#) to aid in this evaluation.

Choose specialized storage

Alternative database solutions often require specific storage solutions. For example, SAP HANA on VMs often employs Azure NetApp Files as its underlying storage solution. Evaluate your vendor's requirements to find an appropriate storage solution to meet your database's requirements. For more information about selecting a storage solution, see [Review your storage options](#).

Criteria for choosing a data store

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article describes the comparison criteria you should use when evaluating a data store. The goal is to help you determine which data storage types can meet your solution's requirements.

General considerations

Keep the following considerations in mind when making your selection.

Functional requirements

- **Data format.** What type of data are you intending to store? Common types include transactional data, JSON objects, telemetry, search indexes, or flat files.
- **Data size.** How large are the entities you need to store? Will these entities need to be maintained as a single document, or can they be split across multiple documents, tables, collections, and so forth?
- **Scale and structure.** What is the overall amount of storage capacity you need? Do you anticipate partitioning your data?
- **Data relationships.** Will your data need to support one-to-many or many-to-many relationships? Are relationships themselves an important part of the data? Will you need to join or otherwise combine data from within the same dataset, or from external datasets?
- **Consistency model.** How important is it for updates made in one node to appear in other nodes, before further changes can be made? Can you accept eventual consistency? Do you need ACID guarantees for transactions?
- **Schema flexibility.** What kind of schemas will you apply to your data? Will you use a fixed schema, a schema-on-write approach, or a schema-on-read approach?
- **Concurrency.** What kind of concurrency mechanism do you want to use when updating and synchronizing data? Will the application perform many updates that could potentially conflict. If so, you may require record locking and pessimistic concurrency control. Alternatively, can you support optimistic concurrency controls? If so, is simple timestamp-based concurrency control enough, or do you need the added functionality of multi-version concurrency control?
- **Data movement.** Will your solution need to perform ETL tasks to move data to other stores or data warehouses?
- **Data lifecycle.** Is the data write-once, read-many? Can it be moved into cool or cold storage?
- **Other supported features.** Do you need any other specific features, such as schema validation, aggregation, indexing, full-text search, MapReduce, or other query capabilities?

Non-functional requirements

- **Performance and scalability.** What are your data performance requirements? Do you have specific requirements for data ingestion rates and data processing rates? What are the acceptable response times for querying and aggregation of data once ingested? How large will you need the data store to scale up? Is your workload more read-heavy or write-heavy?
- **Reliability.** What overall SLA do you need to support? What level of fault-tolerance do you need to provide for data consumers? What kind of backup and restore capabilities do you need?

- **Replication.** Will your data need to be distributed among multiple replicas or regions? What kind of data replication capabilities do you require?
- **Limits.** Will the limits of a particular data store support your requirements for scale, number of connections, and throughput?

Management and cost

- **Managed service.** When possible, use a managed data service, unless you require specific capabilities that can only be found in an IaaS-hosted data store.
- **Region availability.** For managed services, is the service available in all Azure regions? Does your solution need to be hosted in certain Azure regions?
- **Portability.** Will your data need to be migrated to on-premises, external datacenters, or other cloud hosting environments?
- **Licensing.** Do you have a preference of a proprietary versus OSS license type? Are there any other external restrictions on what type of license you can use?
- **Overall cost.** What is the overall cost of using the service within your solution? How many instances will need to run, to support your uptime and throughput requirements? Consider operations costs in this calculation. One reason to prefer managed services is the reduced operational cost.
- **Cost effectiveness.** Can you partition your data, to store it more cost effectively? For example, can you move large objects out of an expensive relational database into an object store?

Security

- **Security.** What type of encryption do you require? Do you need encryption at rest? What authentication mechanism do you want to use to connect to your data?
- **Auditing.** What kind of audit log do you need to generate?
- **Networking requirements.** Do you need to restrict or otherwise manage access to your data from other network resources? Does data need to be accessible only from inside the Azure environment? Does the data need to be accessible from specific IP addresses or subnets? Does it need to be accessible from applications or services hosted on-premises or in other external datacenters?

DevOps

- **Skill set.** Are there particular programming languages, operating systems, or other technology that your team is particularly adept at using? Are there others that would be difficult for your team to work with?
- **Clients** Is there good client support for your development languages?

Choose a data pipeline orchestration technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

Most big data solutions consist of repeated data processing operations, encapsulated in workflows. A pipeline orchestrator is a tool that helps to automate these workflows. An orchestrator can schedule jobs, execute workflows, and coordinate dependencies among tasks.

What are your options for data pipeline orchestration?

In Azure, the following services and tools will meet the core requirements for pipeline orchestration, control flow, and data movement:

- [Azure Data Factory](#)
- [Oozie on HDInsight](#)
- [SQL Server Integration Services \(SSIS\)](#)

These services and tools can be used independently from one another, or used together to create a hybrid solution. For example, the Integration Runtime (IR) in Azure Data Factory V2 can natively execute SSIS packages in a managed Azure compute environment. While there is some overlap in functionality between these services, there are a few key differences.

Key Selection Criteria

To narrow the choices, start by answering these questions:

- Do you need big data capabilities for moving and transforming your data? Usually this means multi-gigabytes to terabytes of data. If yes, then narrow your options to those that best suited for big data.
- Do you require a managed service that can operate at scale? If yes, select one of the cloud-based services that aren't limited by your local processing power.
- Are some of your data sources located on-premises? If yes, look for options that can work with both cloud and on-premises data sources or destinations.
- Is your source data stored in Blob storage on an HDFS filesystem? If so, choose an option that supports Hive queries.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE DATA FACTORY	SQL SERVER INTEGRATION SERVICES (SSIS)	OOZIE ON HDINSIGHT
Managed	Yes	No	Yes
Cloud-based	Yes	No (local)	Yes

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Prerequisite	Azure Subscription	SQL Server	Azure Subscription, HDInsight cluster
Management tools	Azure Portal, PowerShell, CLI, .NET SDK	SSMS, PowerShell	Bash shell, Oozie REST API, Oozie web UI
Pricing	Pay per usage	Licensing / pay for features	No additional charge on top of running the HDInsight cluster

Pipeline capabilities

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Copy data	Yes	Yes	Yes
Custom transformations	Yes	Yes	Yes (MapReduce, Pig, and Hive jobs)
Azure Machine Learning scoring	Yes	Yes (with scripting)	No
HDInsight On-Demand	Yes	No	No
Azure Batch	Yes	No	No
Pig, Hive, MapReduce	Yes	No	Yes
Spark	Yes	No	No
Execute SSIS Package	Yes	Yes	No
Control flow	Yes	Yes	Yes
Access on-premises data	Yes	Yes	No

Scalability capabilities

Capability	Azure Data Factory	SQL Server Integration Services (SSIS)	Oozie on HDInsight
Scale up	Yes	No	No
Scale out	Yes	No	Yes (by adding worker nodes to cluster)
Optimized for big data	Yes	No	Yes

Choose a real-time message ingestion technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

Real time processing deals with streams of data that are captured in real-time and processed with minimal latency. Many real-time processing solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics.

What are your options for real-time message ingestion?

- [Azure Event Hubs](#)
- [Azure IoT Hub](#)
- [Kafka on HDInsight](#)

Azure Event Hubs

[Azure Event Hubs](#) is a highly scalable data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. Event Hubs provides publish-subscribe capabilities with low latency at massive scale, which makes it appropriate for big data scenarios.

Azure IoT Hub

[Azure IoT Hub](#) is a managed service that enables reliable and secure bidirectional communications between millions of IoT devices and a cloud-based back end.

Features of IoT Hub include:

- Multiple options for device-to-cloud and cloud-to-device communication. These options include one-way messaging, file transfer, and request-reply methods.
- Message routing to other Azure services.
- Queryable store for device metadata and synchronized state information.
- Secure communications and access control using per-device security keys or X.509 certificates.
- Monitoring of device connectivity and device identity management events.

In terms of message ingestion, IoT Hub is similar to Event Hubs. However, it was specifically designed for managing IoT device connectivity, not just message ingestion. For more information, see [Comparison of Azure IoT Hub and Azure Event Hubs](#).

Kafka on HDInsight

[Apache Kafka](#) is an open-source distributed streaming platform that can be used to build real-time data pipelines and streaming applications. Kafka also provides message broker functionality similar to a message queue, where you can publish and subscribe to named data streams. It is horizontally scalable, fault-tolerant, and extremely fast. [Kafka on HDInsight](#) provides a Kafka as a managed, highly scalable, and highly available service in Azure.

Some common use cases for Kafka are:

- **Messaging.** Because it supports the publish-subscribe message pattern, Kafka is often used as a message broker.
- **Activity tracking.** Because Kafka provides in-order logging of records, it can be used to track and re-create activities, such as user actions on a web site.
- **Aggregation.** Using stream processing, you can aggregate information from different streams to combine and centralize the information into operational data.
- **Transformation.** Using stream processing, you can combine and enrich data from multiple input topics into one or more output topics.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need two-way communication between your IoT devices and Azure? If so, choose IoT Hub.
- Do you need to manage access for individual devices and be able to revoke access to a specific device? If yes, choose IoT Hub.

Capability matrix

The following tables summarize the key differences in capabilities.

CAPABILITY	IOT HUB	EVENT HUBS	KAFKA ON HDINSIGHT
Cloud-to-device communications	Yes	No	No
Device-initiated file upload	Yes	No	No
Device state information	Device twins	No	No
Protocol support	MQTT, AMQP, HTTPS ¹	AMQP, HTTPS, Kafka Protocol	Kafka Protocol
Security	Per-device identity; revocable access control.	Shared access policies; limited revocation through publisher policies.	Authentication using SASL; pluggable authorization; integration with external authentication services supported.

[1] You can also use [Azure IoT protocol gateway](#) as a custom gateway to enable protocol adaptation for IoT Hub.

For more information, see [Comparison of Azure IoT Hub and Azure Event Hubs](#).

Choose a search data store in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article compares technology choices for search data stores in Azure. A search data store is used to create and store specialized indexes for performing searches on free-form text. The text that is indexed may reside in a separate data store, such as blob storage. An application submits a query to the search data store, and the result is a list of matching documents. For more information about this scenario, see [Processing free-form text for search](#).

What are your options when choosing a search data store?

In Azure, all of the following data stores will meet the core requirements for search against free-form text data by providing a search index:

- [Azure Cognitive Search](#)
- [Elasticsearch](#)
- [HDInsight with Solr](#)
- [Azure SQL Database with full text search](#)

Key selection criteria

For search scenarios, begin choosing the appropriate search data store for your needs by answering these questions:

- Do you want a managed service rather than managing your own servers?
- Can you specify your index schema at design time? If not, choose an option that supports updateable schemas.
- Do you need an index only for full-text search, or do you also need rapid aggregation of numeric data and other analytics? If you need functionality beyond full-text search, consider options that support additional analytics.
- Do you need a search index for log analytics, with support for log collection, aggregation, and visualizations on indexed data? If so, consider Elasticsearch, which is part of a log analytics stack.
- Do you need to index data in common document formats such as PDF, Word, PowerPoint, and Excel? If yes, choose an option that provides document indexers.
- Does your database have specific security needs? If yes, consider the security features listed below.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

Capability	Cognitive Search	Elasticsearch	HDInsight with Solr	SQL Database
Is managed service	Yes	No	Yes	Yes

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
REST API	Yes	Yes	Yes	No
Programmability	.NET, Java, Python, JavaScript	Java	Java	T-SQL
Document indexers for common file types (PDF, DOCX, TXT, and so on)	Yes	No	Yes	No

Manageability capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Updateable schema	Yes	Yes	Yes	Yes
Supports scale out	Yes	Yes	Yes	No

Analytic workload capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Supports analytics beyond full text search	No	Yes	Yes	Yes
Part of a log analytics stack	No	Yes (ELK)	No	No
Supports semantic search	Yes (find similar documents only)	Yes	Yes	Yes

Security capabilities

Capability	Cognitive Search	Elasticsearch	HDIgnition with Solr	SQL Database
Row-level security	Partial (requires application query to filter by group id)	Partial (requires application query to filter by group id)	Yes	Yes
Transparent data encryption	No	No	No	Yes
Restrict access to specific IP addresses	Yes	Yes	Yes	Yes
Restrict access to allow virtual network access only	Yes	Yes	Yes	Yes

CAPABILITY	COGNITIVE SEARCH	ELASTICSEARCH	HDINSIGHT WITH SOLR	SQL DATABASE
Active Directory authentication (integrated authentication)	No	No	No	Yes

See also

[Processing free-form text for search](#)

Choose a stream processing technology in Azure

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article compares technology choices for real-time stream processing in Azure.

Real-time stream processing consumes messages from either queue or file-based storage, processes the messages, and forwards the result to another message queue, file store, or database. Processing may include querying, filtering, and aggregating messages. Stream processing engines must be able to consume endless streams of data and produce results with minimal latency. For more information, see [Real time processing](#).

What are your options when choosing a technology for real-time processing?

In Azure, all of the following data stores will meet the core requirements supporting real-time processing:

- [Azure Stream Analytics](#)
- [HDInsight with Spark Streaming](#)
- [Apache Spark in Azure Databricks](#)
- [HDInsight with Storm](#)
- [Azure Functions](#)
- [Azure App Service WebJobs](#)
- [Apache Kafka streams API](#)

Key Selection Criteria

For real-time processing scenarios, begin choosing the appropriate service for your needs by answering these questions:

- Do you prefer a declarative or imperative approach to authoring stream processing logic?
- Do you need built-in support for temporal processing or windowing?
- Does your data arrive in formats besides Avro, JSON, or CSV? If yes, consider options that support any format using custom code.
- Do you need to scale your processing beyond 1 GB/s? If yes, consider the options that scale with the cluster size.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	AZURE STREAM ANALYTICS	HDINSIGHT WITH SPARK STREAMING	APACHE SPARK IN AZURE DATABRICKS	HDINSIGHT WITH STORM	AZURE FUNCTIONS	AZURE APP SERVICE WEBJOBS

Capability	Azure Stream Analytics	HdInsight with Spark Streaming	Apache Spark in Azure Databricks	HdInsight with Storm	Azure Functions	Azure App Service WebJobs
Programmability	Stream analytics query language, JavaScript	C#/F#, Java, Python, Scala	C#/F#, Java, Python, R, Scala	C#, Java	C#, F#, Java, Node.js, Python	C#, Java, Node.js, PHP, Python
Programming paradigm	Declarative	Mixture of declarative and imperative	Mixture of declarative and imperative	Imperative	Imperative	Imperative
Pricing model	Streaming units	Per cluster hour	Databricks units	Per cluster hour	Per function execution and resource consumption	Per app service plan hour

Integration capabilities

Capability	Azure Stream Analytics	HdInsight with Spark Streaming	Apache Spark in Azure Databricks	HdInsight with Storm	Azure Functions	Azure App Service WebJobs
Inputs	Azure Event Hubs, Azure IoT Hub, Azure Blob storage	Event Hubs, IoT Hub, Kafka, HDFS, Storage Blobs, Azure Data Lake Store	Event Hubs, IoT Hub, Kafka, HDFS, Storage Blobs, Azure Data Lake Store	Event Hubs, IoT Hub, Storage Blobs, Azure Data Lake Store	Supported bindings	Service Bus, Storage Queues, Storage Blobs, Event Hubs, WebHooks, Cosmos DB, Files
Sinks	Azure Data Lake Store, Azure SQL Database, Storage Blobs, Event Hubs, Power BI, Table Storage, Service Bus Queues, Service Bus Topics, Cosmos DB, Azure Functions	HDFS, Kafka, Storage Blobs, Azure Data Lake Store, Cosmos DB	HDFS, Kafka, Storage Blobs, Azure Data Lake Store, Cosmos DB	Event Hubs, Service Bus, Kafka	Supported bindings	Service Bus, Storage Queues, Storage Blobs, Event Hubs, WebHooks, Cosmos DB, Files

Processing capabilities

Capability	Azure Stream Analytics	HDIInsight with Spark Streaming	Apache Spark in Azure Databricks	HDIInsight with Storm	Azure Functions	Azure App Service WebJobs
Built-in temporal/windowing support	Yes	Yes	Yes	Yes	No	No
Input data formats	Avro, JSON or CSV, UTF-8 encoded	Any format using custom code	Any format using custom code	Any format using custom code	Any format using custom code	Any format using custom code
Scalability	Query partitions	Bounded by cluster size	Bounded by Databricks cluster scale configuration	Bounded by cluster size	Up to 200 function app instances processing in parallel	Bounded by app service plan capacity
Late arrival and out of order event handling support	Yes	Yes	Yes	Yes	No	No

See also:

- [Choosing a real-time message ingestion technology](#)
- [Real time processing](#)

Data management patterns

3/10/2022 • 2 minutes to read • [Edit Online](#)

Data management is the key element of cloud applications, and influences most of the quality attributes. Data is typically hosted in different locations and across multiple servers for reasons such as performance, scalability or availability, and this can present a range of challenges. For example, data consistency must be maintained, and data will typically need to be synchronized across different locations.

Additionally data should be protected at rest, in transit, and via authorized access mechanisms to maintain security assurances of confidentiality, integrity, and availability. Refer to the Azure Security Benchmark [Data Protection Control](#) for more information.

PATTERN	SUMMARY
Cache-Aside	Load data on demand into a cache from a data store
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Event Sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.
Index Table	Create indexes over the fields in data stores that are frequently referenced by queries.
Materialized View	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for required query operations.
Sharding	Divide a data store into a set of horizontal partitions or shards.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.
Valet Key	Use a token or key that provides clients with restricted direct access to a specific resource or service.

Data management in the retail industry

3/10/2022 • 11 minutes to read • [Edit Online](#)

Introduction

Data is the foundation for developing and delivering better retail experiences. Data is found in every facet of a retail organization and can be used to extract insights across the value chain into operational performance and customer behavior, as well as leveraged to power improved service experiences. From online browsing to social engagement to in-store purchasing, data abounds. However, capturing data is only a portion of data management. Stitching together disparate data for analysis requires proper handling of data across an organization—thus improving a retailer's ability to make impactful decisions about running their business.

For example, with the growth of mobile shopping, customers have come to expect that retailers have a reasonable amount of data about their shopping habits to be used to improve the experience. A use case example is a personalized product and promotion offering sent directly to a customer's mobile device when shopping in a specific location within a physical retail store. Leveraging data on what, where, how, how many and how often, plus additional inputs such as store product availability, creates opportunities to send real-time promotion messages to a customer's device when the customer is shopping in proximity of a targeted product.

Effective data usage can activate the customer to buy by helping the retailer delivering a more relevant experience; for example, retailers might send the customer a notification with a discount code for the retailer's eCommerce website. Further, this data will drive actionable insights from which company leaders may steer their actions with data-backed decisions

The action to offer a promotion is informed by a combination of data points and triggered by the customer entering the store. The ability to make these connections and the resulting actions are based on the data management model shown below.



Figure 1

When bringing data into Azure, consider the 3Ps of data sources and their applicability to the scenarios the retailer wants to enable. The 3Ps of data sources are Purchased, Public, and Proprietary.

Purchased data typically augments and enhances the organization's existing data most often with market and demographic data that supplements the organization's data capture reach. For example, a retailer may purchase additional demographic data to augment a master customer record, ensuring the record is accurate and complete.

Public data is freely available and may be harvested from social media, government resources (e.g. geography), and other online sources. This data can infer insights such as weather patterns that correlate with purchasing patterns or social engagement that signals product popularity amongst a specific geography. Public data is often available via APIs.

Proprietary data resides within the organization. It may be a retailer's on-premises systems, SaaS applications, or cloud providers. To access the data in a SaaS application provider, and other vendor data, APIs are typically used to communicate the vendor's system. This includes data such as eCommerce site logs, POS sales data, and inventory management systems.

These different data types are used for various insights coming from the data management pipeline.

Ingest

Initially, data is loaded into Azure in its native format, and is stored accordingly. Receiving and managing disparate data sources can be daunting, but Microsoft Azure offers services to load data into the cloud quickly and easily, making it available for processing in the data management pipeline.

Azure has several helpful services for migrating data. The choice depends on the type of data being migrated. [Azure Data Migration Services](#) for SQL Server and the [Azure Import/Export Service](#) are services to help get data into Azure. Other data ingress services to consider include [Azure Data Factory](#) and [Azure Logic Apps](#) connectors. Each has its own features and should be investigated to see which technology works best for the given situation.

Data ingestion isn't limited to Microsoft technologies. Through the [Azure Marketplace](#), retailers may configure many different vendor databases in Azure to work with existing on-premises systems.

Not all data must be maintained in Azure. For example, Point of Sale (POS) data may be held on-premises so Internet outages do not impact sales transactions. This data can be queued and uploaded to Azure on a schedule (perhaps nightly or weekly) for use in analysis, but always treating the on-premises data as the source of truth.

Prepare

Before analysis begins, the data must be prepared. This shaping of data is important to ensure quality of predictive models, reporting KPIs and relevancy of data.

There are two types of data to address when preparing data for analysis, structured and unstructured. Structured data is easier to deal with since it is already formed and formatted. It may require just a simple transformation to go from structured data in source format to structured data which is ready for analysis jobs. Unstructured data typically provides more challenges. Unstructured data isn't stored in a fixed record length format. Examples include documents, social media feeds, and digital images and videos. These data must be managed differently than structured data and often require a dedicated process to ensure these data end up in the right data store, in a useable way.

Data shaping occurs during the Extract-Transform-Load (ETL) process, in the preparation stage. Data is extracted from the unchanged data sources imported into Azure, "cleaned" or reformatted as needed, and stored in a new, more structured format. A common ETL data preparation operation is to transform .csv or Excel files into parquet files, which are easier for machine learning systems like Apache Spark to read and process quickly. Another common scenario is to create XML files or JSON from .csv files, or other formats. The resulting format is easier to use with other analysis engines.

In Azure, there are several transformation technologies available as a ETL services to reshape data. Options include [Azure Databricks](#), [Azure Functions](#) or Logic Apps. Databricks is a fully managed instance of Apache Spark, and is used to transform data from one form to another. Azure Functions are stateless (or "serverless") functions with triggers to fire them and run code. Logic Apps integrates services.

Store

Storing data before processing requires consideration. Data can come in structured or unstructured formats and the shape of the data often determines its storage destination. For example, highly structured data may be suitable for Azure SQL. Less structured data may be held in blob storage, file storage, or table storage.

Data stored in Azure has great performance backed up by a solid service-level agreement (SLA). Data services provide easier to manage solutions, high availability, replication across multiple geographic locations and—above all—Azure offers the data stores and services needed to drive Machine Learning.

Both structured and unstructured data can be stored in [Azure Data Lake](#) and queried using [U-SQL](#), a query

language specific to Azure Data Lake. Examples of data that may be included in a Data Lake include the following, which are divided into commonly structured and unstructured data sources.

Structured data

- CRM data and other line of business applications
- POS transaction data
- Sensor data
- Relational data
- eCommerce transaction data

Unstructured data

- Social feeds
- Video
- Digital images
- Website clickstream analysis

There are a growing number of use cases supporting unstructured data to generate value. This is propelled by the desire for data-driven decisions and the advancement in technology such as AI to enable capture and processing of data at scale. For example, data can include photos or streaming video. For example, streaming video can be leveraged to detect customer shopping selections for a seamless checkout; or product catalog data can be merged seamlessly with a customer's photo of their favorite dress to provide a view of similar, or recommended items.

Examples of structured data include relational database data feeds, sensor data, Apache Parquet files, and ecommerce data. The inherent structure of these data makes them well-suited for a Machine Learning pipeline.

Azure Data Lake service also enables batch and interactive queries along with real time analytics using [Data Lake Analytics](#). Also, Data Lake is specifically well-suited for very large data analysis workloads. Finally, data in the Data Lake is persistent and has no time limit.

Other data stores such as relational databases, Blob storage, Azure Files storage, and Cosmos DB document storage may also hold clean data ready for downstream analysis in the data management pipeline. There is no requirement that one uses a Data Lake.

Analyze

For problems like reducing cost of inventory, retailors can use analysis performed by a Machine Learning process.

Data analysis prepares data for processing through a Machine Learning engine to gain deeper insights into the customer experience. This process produces a model that "learns" and may be applied to future data to predict outcomes. Models define the data that will be examined and how the data will be analyzed through various algorithms. Using the output data from the analysis with data visualization is what could trigger an insight—such as offering an in-store coupon for an item from the customer's wish list in the retailors eCommerce platform.

Data analysis occurs by feeding learning ecosystems with data stored for processing. Typically, this is machine learning performed by Hadoop, Databricks, or a self-managed Spark instance running on a virtual machine. This can also be done simply by querying for data. Insight into KPIs can often be found in clean data without going through a machine learning pipeline.

[Hadoop](#) is part of the fully managed Azure service, [HDInsight](#). HDInsight is a collection of data learning tools used for training data models, outputting data to a data warehouse, and performing queries on Hadoop through the Hive query language. HDInsight can analyze streaming or historical data.

A variety of learning algorithms may be applied to the data as part of training and to maintain data models. A data model explicitly determines the structure of data produced for analysts.

First, the data is cleaned and formed appropriately. It is then processed by a machine learning system such as HDInsight or Apache Spark. To do this, existing data is used to train a model, which in turn is used in analysis of data. The trained model is updated periodically with new known good data to increase its accuracy during analysis. Machine learning services use the model to perform an analysis of the data being processed.

After model training and running a data analysis process, data derived from machine learning analysis can be stored in a data warehouse, or normalized storage databases for analytics data. Microsoft provides [Power BI](#), a fully featured data analytics tool, for deep analysis of data in the data warehouse.

Action

Data in retail moves constantly, and systems that handle it must do so in a timely manner. For example, eCommerce shopper data needs to be processed quickly. This is so items in a buyer's cart can be used to offer additional services, or add-on items during the checkout process. This form of data handling and analysis must occur almost immediately and is typically carried out by systems performing "micro-batch" transactions. That is, data is analyzed in a system which has access to already processed data and is run through a model.

Other "batch" operations may occur at regular intervals but need not occur in near real time. When batch analysis occurs on-premises, these jobs often run at night, on weekends, or when resources are not in use. With Azure, scaling large batch jobs and the virtual machines needed to support them may occur at any time.

Use the following steps to get started.

1. Create a data ingestion plan for data stores providing value to the analysis to be performed. With a detailed data synchronization or migration plan in place, get the data into Azure in its original format.
2. Determine the actionable insights needed and choose a data processing pipeline to accommodate the data processing activities.
3. With these data features in mind, create a data processing pipeline using the appropriate algorithms to gain the insights being sought.
4. Use a common data model for output into a data warehouse, if possible; this can expose the most interesting data features. This usually means reading data in the original Azure storage systems and writing the cleaned version to another data store.
5. Process the data through the machine learning pipelines provided by Spark or Hadoop. Then feed the output to a data warehouse. There are many default algorithms to process the data, or retailers can implement their own. In addition to ML scenarios, load data into standard data storage and enforce a common data model, then query for KPI data. For example, data may be stored in a star schema or other data store.

With data now ready to be used by data analysts, actionable insights may be discovered, and action taken to exploit this new knowledge. For example, a customer's purchase preferences may be loaded back into the retailer's systems and used to improve several customer touchpoints such as the following.

- Increase the average eCommerce or POS transaction by bundling products
- Purchase history in CRM to support customer call center inquiries
- Product suggestions tailored by an e-commerce recommendation engine
- Targeted and relevant ads based on customer data
- Updated inventory availability based on product movement within the supply chain

Another type of insight that may arise are patterns not previously questioned. For example, it may be discovered that more inventory loss happens between the hours of 3:00 PM and 7:00 PM. This might imply the need for

additional data to determine a root cause and a course of action—such as improved security or standard operating procedures.

Conclusion

Data management in retail is complex. But it offers the valuable ability to deliver relevance and an improved customer experience. Using the techniques in this article, insights may be gained to improve the customer experience, drive profitable business outcomes and uncover trends that may drive operational improvements.

Contributors

This article is being updated and maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [David Starr](#) | Principal Solutions Architect
- [Mariya Zorotovich](#) | Head of Customer Experience, HLS & Emerging Technology

Next steps

To continue to understand more of Azure capabilities related to implementing a data management pipeline, read the following:

- See how [Azure Data Factory](#) can help ingest data from on-premises data stores into Azure.
- Learn more about how [Azure Data Lake](#) can serve as a store all data, both structured and unstructured.
- See actual retail reports illustrating how [Power BI](#) can give deeper insights into known questions, but enable trend analysis.
- Visit the [Azure Marketplace](#) to find solutions compatible with those already on-premises.

Transfer data to and from Azure

3/10/2022 • 7 minutes to read • [Edit Online](#)

There are several options for transferring data to and from Azure, depending on your needs.

Physical transfer

Using physical hardware to transfer data to Azure is a good option when:

- Your network is slow or unreliable.
- Getting additional network bandwidth is cost-prohibitive.
- Security or organizational policies do not allow outbound connections when dealing with sensitive data.

If your primary concern is how long it will take to transfer your data, you may want to run a test to verify whether network transfer is actually slower than physical transport.

There are two main options for physically transporting data to Azure:

- **Azure Import/Export.** The [Azure Import/Export service](#) lets you securely transfer large amounts of data to Azure Blob Storage or Azure Files by shipping internal SATA HDDs or SSDs to an Azure datacenter. You can also use this service to transfer data from Azure Storage to hard disk drives and have these shipped to you for loading on-premises.
- **Azure Data Box.** [Azure Data Box](#) is a Microsoft-provided appliance that works much like the Azure Import/Export service. Microsoft ships you a proprietary, secure, and tamper-resistant transfer appliance and handles the end-to-end logistics, which you can track through the portal. One benefit of the Azure Data Box service is ease of use. You don't need to purchase several hard drives, prepare them, and transfer files to each one. Azure Data Box is supported by a number of industry-leading Azure partners to make it easier to seamlessly use offline transport to the cloud from their products.

Command line tools and APIs

Consider these options when you want scripted and programmatic data transfer.

- **Azure CLI.** The [Azure CLI](#) is a cross-platform tool that allows you to manage Azure services and upload data to Azure Storage.
- **AzCopy.** Use AzCopy from a [Windows](#) or [Linux](#) command-line to easily copy data to and from Azure Blob, File, and Table storage with optimal performance. AzCopy supports concurrency and parallelism, and the ability to resume copy operations when interrupted. You can also use AzCopy to copy data from AWS to Azure. For programmatic access, the [Microsoft Azure Storage Data Movement Library](#) is the core framework that powers AzCopy. It is provided as a .NET Core library.
- **PowerShell.** The `Start-AzureStorageBlobCopy` [PowerShell cmdlet](#) is an option for Windows administrators who are used to PowerShell.
- **AdlCopy.** [AdlCopy](#) enables you to copy data from Azure Storage Blobs into Data Lake Store. It can also be used to copy data between two Azure Data Lake Store accounts. However, it cannot be used to copy data from Data Lake Store to Storage Blobs.
- **Distcp.** If you have an HDInsight cluster with access to Data Lake Store, you can use Hadoop ecosystem tools like [Distcp](#) to copy data to and from an HDInsight cluster storage (WASB) into a Data Lake Store account.

- **Sqoop.** [Sqoop](#) is an Apache project and part of the Hadoop ecosystem. It comes preinstalled on all HDInsight clusters. It allows data transfer between an HDInsight cluster and relational databases such as SQL, Oracle, MySQL, and so on. Sqoop is a collection of related tools, including import and export. Sqoop works with HDInsight clusters using either Azure Storage blobs or Data Lake Store attached storage.
- **PolyBase.** [PolyBase](#) is a technology that accesses data outside of the database through the T-SQL language. In SQL Server 2016, it allows you to run queries on external data in Hadoop or to import/export data from Azure Blob Storage. In Azure Synapse Analytics, you can import/export data from Azure Blob Storage and Azure Data Lake Store. Currently, PolyBase is the fastest method of importing data into Azure Synapse.
- **Hadoop command line.** When you have data that resides on an HDInsight cluster head node, you can use the `hadoop -copyFromLocal` command to copy that data to your cluster's attached storage, such as Azure Storage blob or Azure Data Lake Store. In order to use the Hadoop command, you must first connect to the head node. Once connected, you can upload a file to storage.

Graphical interface

Consider the following options if you are only transferring a few files or data objects and don't need to automate the process.

- **Azure Storage Explorer.** [Azure Storage Explorer](#) is a cross-platform tool that lets you manage the contents of your Azure storage accounts. It allows you to upload, download, and manage blobs, files, queues, tables, and Azure Cosmos DB entities. Use it with Blob storage to manage blobs and folders, as well as upload and download blobs between your local file system and Blob storage, or between storage accounts.
- **Azure portal.** Both Blob storage and Data Lake Store provide a web-based interface for exploring files and uploading new files one at a time. This is a good option if you do not want to install any tools or issue commands to quickly explore your files, or to simply upload a handful of new ones.

Data pipeline

Azure Data Factory. [Azure Data Factory](#) is a managed service best suited for regularly transferring files between a number of Azure services, on-premises, or a combination of the two. Using Azure Data Factory, you can create and schedule data-driven workflows (called pipelines) that ingest data from disparate data stores. It can process and transform the data by using compute services such as Azure HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Azure Machine Learning. Create data-driven workflows for [orchestrating](#) and automating data movement and data transformation.

Key Selection Criteria

For data transfer scenarios, choose the appropriate system for your needs by answering these questions:

- Do you need to transfer very large amounts of data, where doing so over an Internet connection would take too long, be unreliable, or too expensive? If yes, consider physical transfer.
- Do you prefer to script your data transfer tasks, so they are reusable? If so, select one of the command line options or Azure Data Factory.
- Do you need to transfer a very large amount of data over a network connection? If so, select an option that is optimized for big data.
- Do you need to transfer data to or from a relational database? If yes, choose an option that supports one or more relational databases. Note that some of these options also require a Hadoop cluster.

- Do you need an automated data pipeline or workflow orchestration? If yes, consider Azure Data Factory.

Capability matrix

The following tables summarize the key differences in capabilities.

Physical transfer

CAPABILITY	AZURE IMPORT/EXPORT SERVICE	AZURE DATA BOX
Form factor	Internal SATA HDDs or SSDs	Secure, tamper-proof, single hardware appliance
Microsoft manages shipping logistics	No	Yes
Integrates with partner products	No	Yes
Custom appliance	No	Yes

Command line tools

Hadoop/HDInsight:

CAPABILITY	DISTCP	SQOOP	HADOOP CLI
Optimized for big data	Yes	Yes	Yes
Copy to relational database	No	Yes	No
Copy from relational database	No	Yes	No
Copy to Blob storage	Yes	Yes	Yes
Copy from Blob storage	Yes	Yes	No
Copy to Data Lake Store	Yes	Yes	Yes
Copy from Data Lake Store	Yes	Yes	No

Other:

CAPABILITY	AZURE CLI	AZCOPY	POWERSHELL	ADLCOPY	POLYBASE
Compatible platforms	Linux, OS X, Windows	Linux, Windows	Windows	Linux, OS X, Windows	SQL Server, Azure Synapse
Optimized for big data	No	Yes	No	Yes ¹	Yes ²
Copy to relational database	No	No	No	No	Yes

CAPABILITY	AZURE CLI	AZCOPY	POWERSHELL	ADLCOPY	POLYBASE
Copy from relational database	No	No	No	No	Yes
Copy to Blob storage	Yes	Yes	Yes	No	Yes
Copy from Blob storage	Yes	Yes	Yes	Yes	Yes
Copy to Data Lake Store	No	Yes	Yes	Yes	Yes
Copy from Data Lake Store	No	No	Yes	Yes	Yes

[1] AdlCopy is optimized for transferring big data when used with a Data Lake Analytics account.

[2] PolyBase [performance can be increased](#) by pushing computation to Hadoop and using [PolyBase scale-out groups](#) to enable parallel data transfer between SQL Server instances and Hadoop nodes.

Graphical interface and Azure Data Factory

CAPABILITY	AZURE STORAGE EXPLORER	AZURE PORTAL *	AZURE DATA FACTORY
Optimized for big data	No	No	Yes
Copy to relational database	No	No	Yes
Copy from relational database	No	No	Yes
Copy to Blob storage	Yes	No	Yes
Copy from Blob storage	Yes	No	Yes
Copy to Data Lake Store	No	No	Yes
Copy from Data Lake Store	No	No	Yes
Upload to Blob storage	Yes	Yes	Yes
Upload to Data Lake Store	Yes	Yes	Yes
Orchestrate data transfers	No	No	Yes
Custom data transformations	No	No	Yes
Pricing model	Free	Free	Pay per usage

* Azure portal in this case means using the web-based exploration tools for Blob storage and Data Lake Store.

Extend on-premises data solutions to the cloud

3/10/2022 • 7 minutes to read • [Edit Online](#)

When organizations move workloads and data to the cloud, their on-premises datacenters often continue to play an important role. The term *hybrid cloud* refers to a combination of public cloud and on-premises datacenters, to create an integrated IT environment that spans both. Some organizations use hybrid cloud as a path to migrate their entire datacenter to the cloud over time. Other organizations use cloud services to extend their existing on-premises infrastructure.

This article describes some considerations and best practices for managing data in a hybrid cloud solution,

When to use a hybrid solution

Consider using a hybrid solution in the following scenarios:

- As a transition strategy during a longer-term migration to a fully cloud-native solution.
- When regulations or policies do not permit moving specific data or workloads to the cloud.
- For disaster recovery and fault tolerance, by replicating data and services between on-premises and cloud environments.
- To reduce latency between your on-premises datacenter and remote locations, by hosting part of your architecture in Azure.

Challenges

- Creating a consistent environment in terms of security, management, and development, and avoiding duplication of work.
- Creating a reliable, low latency and secure data connection between your on-premises and cloud environments.
- Replicating your data and modifying applications and tools to use the correct data stores within each environment.
- Securing and encrypting data that is hosted in the cloud but accessed from on-premises, or vice versa.

On-premises data stores

On-premises data stores include databases and files. There may be several reasons to keep these data stores local. There may be regulations or policies that do not permit moving specific data or workloads to the cloud. Data sovereignty, privacy, or security concerns may favor on-premises placement. During a migration, you may want to keep some data local to an application that hasn't been migrated yet.

Considerations in placing application data in a public cloud include:

- **Cost.** The cost of storage in Azure can be significantly lower than the cost of maintaining storage with similar characteristics in an on-premises datacenter. Of course, many companies have existing investments in high-end SANs, so these cost advantages may not reach full fruition until existing hardware ages out.
- **Elastic scale.** Planning and managing data capacity growth in an on-premises environment can be challenging, particularly when data growth is difficult to predict. These applications can take advantage of the capacity-on-demand and virtually unlimited storage available in the cloud. This consideration is less

relevant for applications that consist of relatively static-sized datasets.

- **Disaster recovery.** Data stored in Azure can be automatically replicated within an Azure region and across geographic regions. In hybrid environments, these same technologies can be used to replicate between on-premises and cloud-based data stores.

Extending data stores to the cloud

There are several options for extending on-premises data stores to the cloud. One option is to have on-premises and cloud replicas. This can help achieve a high level of fault tolerance, but may require making changes to applications to connect to the appropriate data store in the event of a failover.

Another option is to move a portion of the data to cloud storage, while keeping the more current or more highly accessed data on-premises. This method can provide a more cost-effective option for long-term storage, as well as improve data access response times by reducing your operational data set.

A third option is to keep all data on-premises, but use cloud computing to host applications. To do this, you would host your application in the cloud and connect it to your on-premises data store over a secure connection.

Azure Stack

For a complete hybrid cloud solution, consider using [Microsoft Azure Stack](#). Azure Stack is a hybrid cloud platform that lets you provide Azure services from your datacenter. This helps maintain consistency between on-premises and Azure, by using identical tools and requiring no code changes.

The following are some use cases for Azure and Azure Stack:

- **Edge and disconnected solutions.** Address latency and connectivity requirements by processing data locally in Azure Stack and then aggregating in Azure for further analytics, with common application logic across both.
- **Cloud applications that meet varied regulations.** Develop and deploy applications in Azure, with the flexibility to deploy the same applications on-premises on Azure Stack to meet regulatory or policy requirements.
- **Cloud application model on-premises.** Use Azure to update and extend existing applications or build new ones. Use consistent DevOps processes across Azure in the cloud and Azure Stack on-premises.

SQL Server data stores

If you are running SQL Server on-premises, you can use Microsoft Azure Blob Storage service for backup and restore. For more information, see [SQL Server Backup and Restore with Microsoft Azure Blob Storage Service](#). This capability gives you limitless offsite storage, and the ability to share the same backups between SQL Server running on-premises and SQL Server running in a virtual machine in Azure.

[Azure SQL Database](#) is a managed relational database-as-a service. Because Azure SQL Database uses the Microsoft SQL Server Engine, applications can access data in the same way with both technologies. Azure SQL Database can also be combined with SQL Server in useful ways. For example, the [SQL Server Stretch Database](#) feature lets an application access what looks like a single table in a SQL Server database while some or all rows of that table might be stored in Azure SQL Database. This technology automatically moves data that's not accessed for a defined period of time to the cloud. Applications reading this data are unaware that any data has been moved to the cloud.

Maintaining data stores on-premises and in the cloud can be challenging when you desire to keep the data synchronized. You can address this with [SQL Data Sync](#), a service built on Azure SQL Database that lets you synchronize the data you select, bi-directionally across multiple Azure SQL databases and SQL Server instances.

While Data Sync makes it easy to keep your data up-to-date across these various data stores, it should not be used for disaster recovery or for migrating from on-premises SQL Server to Azure SQL Database.

For disaster recovery and business continuity, you can use [AlwaysOn Availability Groups](#) to replicate data across two or more instances of SQL Server, some of which can be running on Azure virtual machines in another geographic region.

Network shares and file-based data stores

In a hybrid cloud architecture, it is common for an organization to keep newer files on-premises while archiving older files to the cloud. This is sometimes called file tiering, where there is seamless access to both sets of files, on-premises and cloud-hosted. This approach helps to minimize network bandwidth usage and access times for newer files, which are likely to be accessed the most often. At the same time, you get the benefits of cloud-based storage for archived data.

Organizations may also wish to move their network shares entirely to the cloud. This would be desirable, for example, if the applications that access them are also located in the cloud. This procedure can be done using [data orchestration](#) tools.

[Azure StorSimple](#) offers the most complete integrated storage solution for managing storage tasks between your on-premises devices and Azure cloud storage. StorSimple is an efficient, cost-effective, and easily manageable storage area network (SAN) solution that eliminates many of the issues and expenses associated with enterprise storage and data protection. It uses the proprietary StorSimple 8000 series device, integrates with cloud services, and provides a set of integrated management tools.

Another way to use on-premises network shares alongside cloud-based file storage is with [Azure Files](#). Azure Files offers fully managed file shares that you can access with the standard [Server Message Block \(SMB\)](#) protocol (sometimes referred to as CIFS). You can mount Azure Files as a file share on your local computer, or use them with existing applications that access local or network share files.

To synchronize file shares in Azure Files with your on-premises Windows Servers, use [Azure File Sync](#). One major benefit of Azure File Sync is the ability to tier files between your on-premises file server and Azure Files. This lets you keep only the newest and most recently accessed files locally.

For more information, see [Deciding when to use Azure Blob storage, Azure Files, or Azure Disks](#).

Hybrid networking

This article focused on hybrid data solutions, but another consideration is how to extend your on-premises network to Azure. For more information about this aspect of hybrid solutions, see:

- [Choose a solution for connecting an on-premises network to Azure](#)
- [Hybrid network reference architectures](#)

Secure data solutions

3/10/2022 • 5 minutes to read • [Edit Online](#)

For many, making data accessible in the cloud, particularly when transitioning from working exclusively in on-premises data stores, can cause some concern around increased accessibility to that data and new ways in which to secure it.

Challenges

- Centralizing the monitoring and analysis of security events stored in numerous logs.
- Implementing encryption and authorization management across your applications and services.
- Ensuring that centralized identity management works across all of your solution components, whether on-premises or in the cloud.

Data protection

The first step to protecting information is identifying what to protect. Develop clear, simple, and well-communicated guidelines to identify, protect, and monitor the most important data assets anywhere they reside. Establish the strongest protection for assets that have a disproportionate impact on the organization's mission or profitability. These are known as high value assets, or HVAs. Perform stringent analysis of HVA lifecycle and security dependencies, and establish appropriate security controls and conditions. Similarly, identify and classify sensitive assets, and define the technologies and processes to automatically apply security controls.

Once the data you need to protect has been identified, consider how you will protect the data *at rest* and data *in transit*.

- **Data at rest:** Data that exists statically on physical media, whether magnetic or optical disk, on premises or in the cloud.
- **Data in transit:** Data while it is being transferred between components, locations, or programs, such as over the network, across a service bus (from on-premises to cloud and vice-versa), or during an input/output process.

To learn more about protecting your data at rest or in transit, see [Azure Data Security and Encryption Best Practices](#).

Access control

Central to protecting your data in the cloud is a combination of identity management and access control. Given the variety and type of cloud services, as well as the rising popularity of [hybrid cloud](#), there are several key practices you should follow when it comes to identity and access control:

- Centralize your identity management.
- Enable Single Sign-On (SSO).
- Deploy password management.
- Enforce multifactor authentication for users.
- Use Azure role-based access control (Azure RBAC).
- Conditional Access Policies should be configured, which enhances the classic concept of user identity with additional properties related to user location, device type, patch level, and so on.
- Control locations where resources are created using Resource Manager.

- Actively monitor for suspicious activities

For more information, see [Azure Identity Management and access control security best practices](#).

Auditing

Beyond the identity and access monitoring previously mentioned, the services and applications that you use in the cloud should be generating security-related events that you can monitor. The primary challenge to monitoring these events is handling the quantities of logs, in order to avoid potential problems or troubleshoot past ones. Cloud-based applications tend to contain many moving parts, most of which generate some level of logging and telemetry. Use centralized monitoring and analysis to help you manage and make sense of the large amount of information.

For more information, see [Azure Logging and Auditing](#).

Securing data solutions in Azure

Encryption

Virtual machines. Use [Azure Disk Encryption](#) to encrypt the attached disks on Windows or Linux VMs. This solution integrates with [Azure Key Vault](#) to control and manage the disk-encryption keys and secrets.

Azure Storage. Use [Azure Storage Service Encryption](#) to automatically encrypt data at rest in Azure Storage. Encryption, decryption, and key management are totally transparent to users. Data can also be secured in transit by using client-side encryption with Azure Key Vault. For more information, see [Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage](#).

SQL Database and Azure Synapse Analytics. Use [Transparent Data Encryption](#) (TDE) to perform real-time encryption and decryption of your databases, associated backups, and transaction log files without requiring any changes to your applications. SQL Database can also use [Always Encrypted](#) to help protect sensitive data at rest on the server, during movement between client and server, and while the data is in use. You can use Azure Key Vault to store your Always Encrypted encryption keys.

Rights management

[Azure Rights Management](#) is a cloud-based service that uses encryption, identity, and authorization policies to secure files and email. It works across multiple devices, such as phones, tablets, and PCs. Information can be protected both within your organization and outside your organization because that protection remains with the data, even when it leaves your organization's boundaries.

Access control

Use [Azure role-based access control \(Azure RBAC\)](#) to restrict access to Azure resources based on user roles. If you are using Active Directory on-premises, you can [synchronize with Azure AD](#) to provide users with a cloud identity based on their on-premises identity.

Use [Conditional access in Azure Active Directory](#) to enforce controls on the access to applications in your environment based on specific conditions. For example, your policy statement could take the form of: *When contractors are trying to access our cloud apps from networks that are not trusted, then block access.*

[Azure AD Privileged Identity Management](#) can help you manage, control, and monitor your users and what sorts of tasks they are performing with their admin privileges. This is an important step to limiting who in your organization can carry out privileged operations in Azure AD, Azure, Microsoft 365, or SaaS apps, as well as monitor their activities.

Network

To protect data in transit, always use SSL/TLS when exchanging data across different locations. Sometimes you need to isolate your entire communication channel between your on-premises and cloud infrastructure by using

either a virtual private network (VPN) or [ExpressRoute](#). For more information, see [Extending on-premises data solutions to the cloud](#).

Use [network security groups](#) to reduce the number of potential attack vectors. A network security group contains a list of security rules that allow or deny inbound or outbound network traffic based on source or destination IP address, port, and protocol.

Use [Virtual Network service endpoints](#) to secure Azure SQL or Azure Storage resources, so that only traffic from your virtual network can access these resources.

VMs within an Azure Virtual Network (VNet) can securely communicate with other VNets using [virtual network peering](#). Network traffic between peered virtual networks is private. Traffic between the virtual networks is kept on the Microsoft backbone network.

For more information, see [Azure network security](#)

Monitoring

[Microsoft Defender for Cloud](#) automatically collects, analyzes, and integrates log data from your Azure resources, the network, and connected partner solutions, such as firewall solutions, to detect real threats and reduce false positives.

[Log Analytics](#) provides centralized access to your logs and helps you analyze that data and create custom alerts.

[Azure SQL Database Threat Detection](#) detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases. Security officers or other designated administrators can receive an immediate notification about suspicious database activities as they occur. Each notification provides details of the suspicious activity and recommends how to further investigate and mitigate the threat.

Monitoring Azure Databricks

3/10/2022 • 2 minutes to read • [Edit Online](#)

Azure Databricks is a fast, powerful Apache Spark–based analytics service that makes it easy to rapidly develop and deploy big data analytics and artificial intelligence (AI) solutions. Many users take advantage of the simplicity of notebooks in their Azure Databricks solutions. For users that require more robust computing options, Azure Databricks supports the distributed execution of custom application code.

Monitoring is a critical part of any production-level solution, and Azure Databricks offers robust functionality for monitoring custom application metrics, streaming query events, and application log messages. Azure Databricks can send this monitoring data to different logging services.

The following articles show how to send monitoring data from Azure Databricks to [Azure Monitor](#), the monitoring data platform for Azure.

- [Send Azure Databricks application logs to Azure Monitor](#)
- [Use dashboards to visualize Azure Databricks metrics](#)
- [Troubleshoot performance bottlenecks](#)

The code library that accompanies these articles extends the core monitoring functionality of Azure Databricks to send Spark metrics, events, and logging information to Azure Monitor.

The audience for these articles and the accompanying code library are Apache Spark and Azure Databricks solution developers. The code must be built into Java Archive (JAR) files and then deployed to an Azure Databricks cluster. The code is a combination of [Scala](#) and Java, with a corresponding set of [Maven](#) project object model (POM) files to build the output JAR files. Understanding of Java, Scala, and Maven are recommended as prerequisites.

Next steps

Start by building the code library and deploying it to your Azure Databricks cluster.

[Send Azure Databricks application logs to Azure Monitor](#)

Send Azure Databricks application logs to Azure Monitor

3/10/2022 • 2 minutes to read • [Edit Online](#)

This article shows how to send application logs and metrics from Azure Databricks to a [Log Analytics workspace](#). It uses the [Azure Databricks Monitoring Library](#), which is available on GitHub.

Prerequisites

Configure your Azure Databricks cluster to use the monitoring library, as described in the [GitHub readme](#).

NOTE

The monitoring library streams Apache Spark level events and Spark Structured Streaming metrics from your jobs to Azure Monitor. You don't need to make any changes to your application code for these events and metrics.

Send application metrics using Dropwizard

Spark uses a configurable metrics system based on the Dropwizard Metrics Library. For more information, see [Metrics](#) in the Spark documentation.

To send application metrics from Azure Databricks application code to Azure Monitor, follow these steps:

1. Build the `spark-listeners-loganalytics-1.0-SNAPSHOT.jar` JAR file as described in the [GitHub readme](#).
2. Create Dropwizard [gauges or counters](#) in your application code. You can use the `UserMetricsSystem` class defined in the monitoring library. The following example creates a counter named `counter1`.

```
import org.apache.spark.metrics.UserMetricsSystems
import org.apache.spark.sql.SparkSession

object StreamingQueryListenerSampleJob {

    private final val METRICS_NAMESPACE = "samplejob"
    private final val COUNTER_NAME = "counter1"

    def main(args: Array[String]): Unit = {

        val spark = SparkSession
            .builder
            .getOrCreate

        val driverMetricsSystem = UserMetricsSystems
            .getMetricSystem(METRICS_NAMESPACE, builder => {
                builder.registerCounter(COUNTER_NAME)
            })

        driverMetricsSystem.counter(COUNTER_NAME).inc(5)
    }
}
```

The monitoring library includes a [sample application](#) that demonstrates how to use the

UserMetricsSystem class.

Send application logs using Log4j

To send your Azure Databricks application logs to Azure Log Analytics using the [Log4j appender](#) in the library, follow these steps:

1. Build the **spark-listeners-1.0-SNAPSHOT.jar** and the **spark-listeners-loganalytics-1.0-SNAPSHOT.jar** JAR file as described in the [GitHub readme](#).
2. Create a **log4j.properties** configuration file for your application. Include the following configuration properties. Substitute your application package name and log level where indicated:

```
log4j.appenders.A1=com.microsoft.pnp.logging.loganalytics.LogAnalyticsAppender
log4j.appenders.A1.layout=com.microsoft.pnp.logging.JSONLayout
log4j.appenders.A1.layout.LocationInfo=false
log4j.additivity.<your application package name>=false
log4j.logger.<your application package name>=<log level>, A1
```

You can find a sample configuration file [here](#).

3. In your application code, include the **spark-listeners-loganalytics** project, and import [com.microsoft.pnp.logging.Log4jConfiguration](#) to your application code.

```
import com.microsoft.pnp.logging.Log4jConfiguration
```

4. Configure Log4j using the **log4j.properties** file you created in step 3:

```
getClass.getResourceAsStream("<path to file in your JAR file>/log4j.properties") {
    stream => {
        Log4jConfiguration.configure(stream)
    }
}
```

5. Add Apache Spark log messages at the appropriate level in your code as required. For example, use the [logDebug](#) method to send a debug log message. For more information, see [Logging in the Spark documentation](#).

```
logTrace("Trace message")
logDebug("Debug message")
logInfo("Info message")
logWarning("Warning message")
logError("Error message")
```

Run the sample application

The monitoring library includes a [sample application](#) that demonstrates how to send both application metrics and application logs to Azure Monitor. To run the sample:

1. Build the **spark-jobs** project in the monitoring library, as described in the [GitHub readme](#).
2. Navigate to your Databricks workspace and create a new job, as described [here](#).
3. In the job detail page, select **Set JAR**.
4. Upload the JAR file from [/src/spark-jobs/target/spark-jobs-1.0-SNAPSHOT.jar](#).

5. For Main class, enter `com.microsoft.pnp.samplejob.StreamingQueryListenerSampleJob`.
6. Select a cluster that is already configured to use the monitoring library. See [Configure Azure Databricks to send metrics to Azure Monitor](#).

When the job runs, you can view the application logs and metrics in your Log Analytics workspace.

Application logs appear under SparkLoggingEvent_CL:

```
SparkLoggingEvent_CL | where logger_name_s contains "com.microsoft.pnp"
```

Application metrics appear under SparkMetric_CL:

```
SparkMetric_CL | where name_s contains "rowcounter" | limit 50
```

IMPORTANT

After you verify the metrics appear, stop the sample application job.

Next steps

Deploy the performance monitoring dashboard that accompanies this code library to troubleshoot performance issues in your production Azure Databricks workloads.

[Use dashboards to visualize Azure Databricks metrics](#)

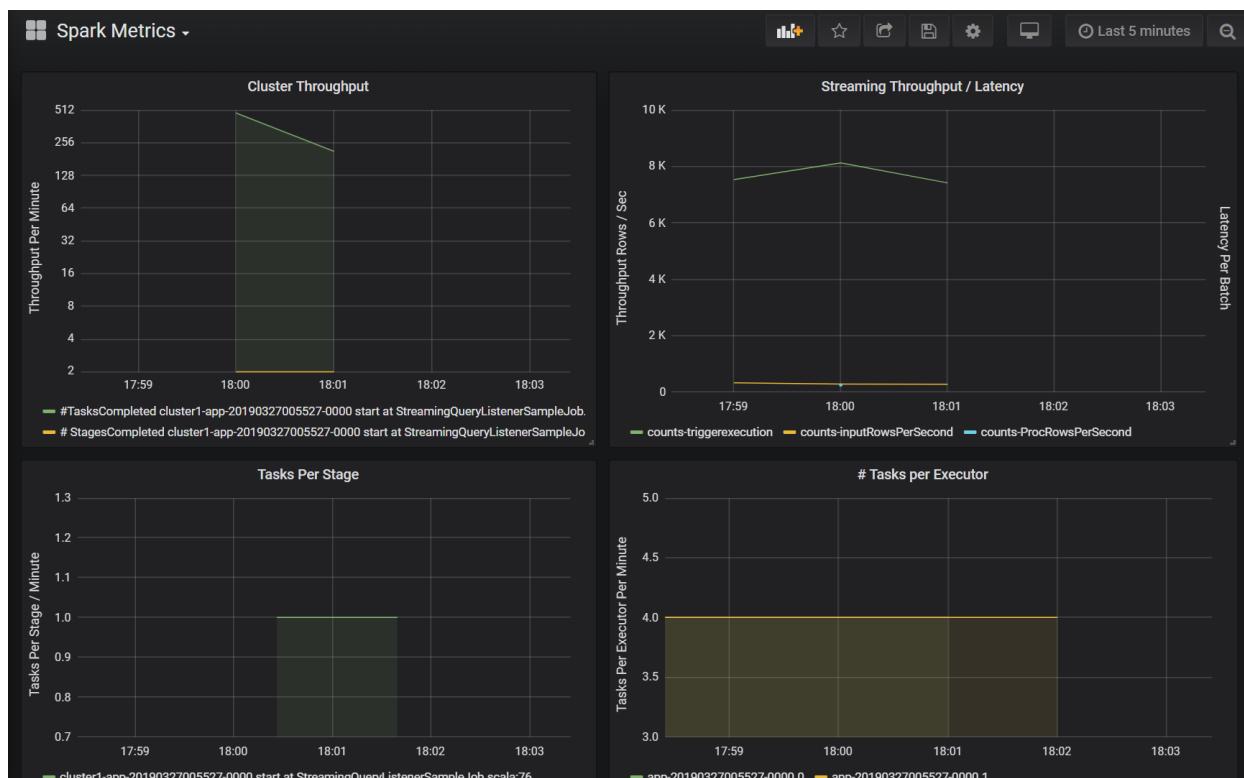
Use dashboards to visualize Azure Databricks metrics

3/10/2022 • 7 minutes to read • [Edit Online](#)

This article shows how to set up a Grafana dashboard to monitor Azure Databricks jobs for performance issues.

Azure Databricks is a fast, powerful, and collaborative Apache Spark–based analytics service that makes it easy to rapidly develop and deploy big data analytics and artificial intelligence (AI) solutions. Monitoring is a critical component of operating Azure Databricks workloads in production. The first step is to gather metrics into a workspace for analysis. In Azure, the best solution for managing log data is Azure Monitor. Azure Databricks does not natively support sending log data to Azure monitor, but a [library for this functionality](#) is available in [GitHub](#).

This library enables logging of Azure Databricks service metrics as well as Apache Spark structure streaming query event metrics. Once you've successfully deployed this library to an Azure Databricks cluster, you can further deploy a set of [Grafana](#) dashboards that you can deploy as part of your production environment.



Prerequisites

Configure your Azure Databricks cluster to use the monitoring library, as described in the [GitHub readme](#).

Deploy the Azure Log Analytics workspace

To deploy the Azure Log Analytics workspace, follow these steps:

1. Navigate to the `/perftools/deployment/loganalytics` directory.
2. Deploy the `logAnalyticsDeploy.json` Azure Resource Manager template. For more information about deploying Resource Manager templates, see [Deploy resources with Resource Manager templates and Azure CLI](#). The template has the following parameters:

- **location**: The region where the Log Analytics workspace and dashboards are deployed.
- **serviceTier**: The workspace pricing tier. See [here](#) for a list of valid values.
- **dataRetention** (optional): The number of days the log data is retained in the Log Analytics workspace. The default value is 30 days. If the pricing tier is **Free**, the data retention must be seven days.
- **workspaceName** (optional): A name for the workspace. If not specified, the template generates a name.

```
az deployment group create --resource-group <resource-group-name> --template-file
logAnalyticsDeploy.json --parameters location='East US' serviceTier='Standalone'
```

This template creates the workspace and also creates a set of predefined queries that are used by dashboard.

Deploy Grafana in a virtual machine

Grafana is an open source project you can deploy to visualize the time series metrics stored in your Azure Log Analytics workspace using the Grafana plugin for Azure Monitor. Grafana executes on a virtual machine (VM) and requires a storage account, virtual network, and other resources. To deploy a virtual machine with the bitnami-certified Grafana image and associated resources, follow these steps:

1. Use the Azure CLI to accept the Azure Marketplace image terms for Grafana.

```
az vm image terms accept --publisher bitnami --offer grafana --plan default
```

2. Navigate to the `/spark-monitoring/perf-tools/deployment/grafana` directory in your local copy of the GitHub repo.
3. Deploy the `grafanaDeploy.json` Resource Manager template as follows:

```
export DATA_SOURCE="https://raw.githubusercontent.com/mspnp/spark-
monitoring/master/perf-tools/deployment/grafana/AzureDataSource.sh"
az deployment group create \
--resource-group <resource-group-name> \
--template-file grafanaDeploy.json \
--parameters adminPass='<vm password>' dataSource=$DATA_SOURCE
```

Once the deployment is complete, the bitnami image of Grafana is installed on the virtual machine.

Update the Grafana password

As part of the setup process, the Grafana installation script outputs a temporary password for the **admin** user. You need this temporary password to sign in. To obtain the temporary password, follow these steps:

1. Log in to the Azure portal.
2. Select the resource group where the resources were deployed.
3. Select the VM where Grafana was installed. If you used the default parameter name in the deployment template, the VM name is prefaced with **sparkmonitoring-vm-grafana**.
4. In the **Support + troubleshooting** section, click **Boot diagnostics** to open the boot diagnostics page.
5. Click **Serial log** on the boot diagnostics page.
6. Search for the following string: "Setting Bitnami application password to".
7. Copy the password to a safe location.

Next, change the Grafana administrator password by following these steps:

1. In the Azure portal, select the VM and click **Overview**.
2. Copy the public IP address.
3. Open a web browser and navigate to the following URL: `http://<IP address>:3000`.
4. At the Grafana login screen, enter **admin** for the user name, and use the Grafana password from the previous steps.
5. Once logged in, select **Configuration** (the gear icon).
6. Select **Server Admin**.
7. On the **Users** tab, select the **admin** login.
8. Update the password.

Create an Azure Monitor data source

1. Create a service principal that allows Grafana to manage access to your Log Analytics workspace. For more information, see [Create an Azure service principal with Azure CLI](#)

```
az ad sp create-for-rbac --name http://<service principal name> \
    --role "Log Analytics Reader" \
    --scopes /subscriptions/mySubscriptionID
```

2. Note the values for `appId`, `password`, and `tenant` in the output from this command:

```
{
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "displayName": "azure-cli-2019-03-27-00-33-39",
  "name": "http://<service principal name>",
  "password": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

3. Log into Grafana as described earlier. Select **Configuration** (the gear icon) and then **Data Sources**.
4. In the **Data Sources** tab, click **Add data source**.
5. Select **Azure Monitor** as the data source type.
6. In the **Settings** section, enter a name for the data source in the **Name** textbox.
7. In the **Azure Monitor API Details** section, enter the following information:
 - Subscription Id: Your Azure subscription ID.
 - Tenant Id: The tenant ID from earlier.
 - Client Id: The value of "`appId`" from earlier.
 - Client Secret: The value of "`password`" from earlier.
8. In the **Azure Log Analytics API Details** section, check the **Same Details as Azure Monitor API** checkbox.
9. Click **Save & Test**. If the Log Analytics data source is correctly configured, a success message is displayed.

Create the dashboard

Create the dashboards in Grafana by following these steps:

1. Navigate to the `/perftools/dashboards/grafana` directory in your local copy of the GitHub repo.
2. Run the following script:

```
export WORKSPACE=<your Azure Log Analytics workspace ID>
export LOGTYPE=SparkListenerEvent_CL

sh DashGen.sh
```

The output from the script is a file named **SparkMonitoringDash.json**.

3. Return to the Grafana dashboard and select **Create** (the plus icon).
4. Select **Import**.
5. Click **Upload json File**.
6. Select the **SparkMonitoringDash.json** file created in step 2.
7. In the **Options** section, under **ALA**, select the Azure Monitor data source created earlier.
8. Click **Import**.

Visualizations in the dashboards

Both the Azure Log Analytics and Grafana dashboards include a set of time-series visualizations. Each graph is time-series plot of metric data related to an Apache Spark [job](#), stages of the job, and tasks that make up each stage.

The visualizations are:

Job latency

This visualization shows execution latency for a job, which is a coarse view on the overall performance of a job. Displays the job execution duration from start to completion. Note that the job start time is not the same as the job submission time. Latency is represented as percentiles (10%, 30%, 50%, 90%) of job execution indexed by cluster ID and application ID.

Stage latency

The visualization shows the latency of each stage per cluster, per application, and per individual stage. This visualization is useful for identifying a particular stage that is running slowly.

Task latency

This visualization shows task execution latency. Latency is represented as a percentile of task execution per cluster, stage name, and application.

Sum Task Execution per host

This visualization shows the sum of task execution latency per host running on a cluster. Viewing task execution latency per host identifies hosts that have much higher overall task latency than other hosts. This may mean that tasks have been inefficiently or unevenly distributed to hosts.

Task metrics

This visualization shows a set of the execution metrics for a given task's execution. These metrics include the size and duration of a data shuffle, duration of serialization and deserialization operations, and others. For the full set of metrics, view the Log Analytics query for the panel. This visualization is useful for understanding the operations that make up a task and identifying resource consumption of each operation. Spikes in the graph represent costly operations that should be investigated.

Cluster throughput

This visualization is a high-level view of work items indexed by cluster and application to represent the amount of work done per cluster and application. It shows the number of jobs, tasks, and stages completed per cluster,

application, and stage in one minute increments.

Streaming Throughput/Latency

This visualization is related to the metrics associated with a structured streaming query. The graph shows the number of input rows per second and the number of rows processed per second. The streaming metrics are also represented per application. These metrics are sent when the OnQueryProgress event is generated as the structured streaming query is processed and the visualization represents streaming latency as the amount of time, in milliseconds, taken to execute a query batch.

Resource consumption per executor

Next is a set of visualizations for the dashboard show the particular type of resource and how it is consumed per executor on each cluster. These visualizations help identify outliers in resource consumption per executor. For example, if the work allocation for a particular executor is skewed, resource consumption will be elevated in relation to other executors running on the cluster. This can be identified by spikes in the resource consumption for an executor.

Executor compute time metrics

Next is a set of visualizations for the dashboard that show the ratio of executor serialize time, deserialize time, CPU time, and Java virtual machine time to overall executor compute time. This demonstrates visually how much each of these four metrics is contributing to overall executor processing.

Shuffle metrics

The final set of visualizations shows the data shuffle metrics associated with a structured streaming query across all executors. These include shuffle bytes read, shuffle bytes written, shuffle memory, and disk usage in queries where the file system is used.

Next steps

[Troubleshoot performance bottlenecks](#)

Troubleshoot performance bottlenecks in Azure Databricks

3/10/2022 • 6 minutes to read • [Edit Online](#)

This article describes how to use monitoring dashboards to find performance bottlenecks in Spark jobs on Azure Databricks.

Azure Databricks is an Apache Spark–based analytics service that makes it easy to rapidly develop and deploy big data analytics. Monitoring and troubleshooting performance issues is a critical when operating production Azure Databricks workloads. To identify common performance issues, it's helpful to use monitoring visualizations based on telemetry data.

Prerequisites

To set up the Grafana dashboards shown in this article:

- Configure your Databricks cluster to send telemetry to a Log Analytics workspace, using the Azure Databricks Monitoring Library. For details, see the [GitHub readme](#).
- Deploy Grafana in a virtual machine. See [Use dashboards to visualize Azure Databricks metrics](#).

The Grafana dashboard that is deployed includes a set of time-series visualizations. Each graph is time-series plot of metrics related to an Apache Spark job, the stages of the job, and tasks that make up each stage.

Azure Databricks performance overview

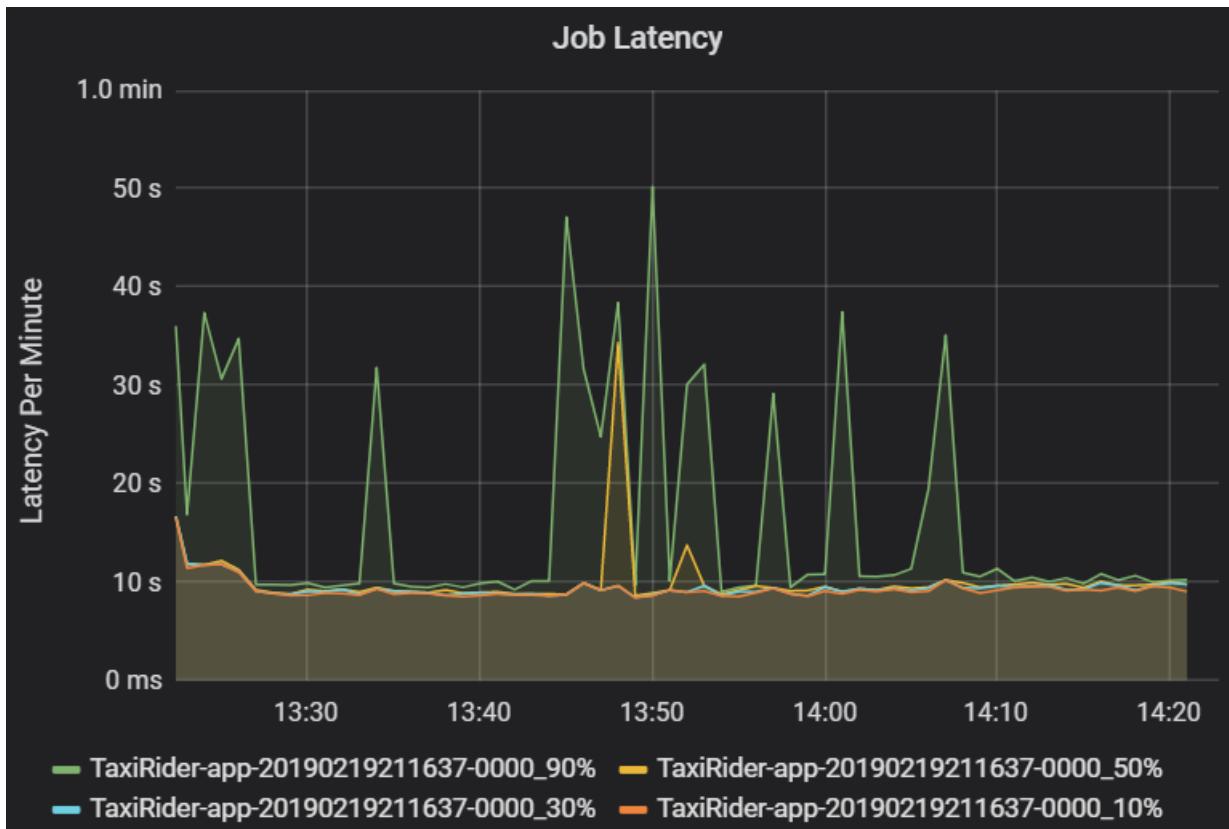
Azure Databricks is based on Apache Spark, a general-purpose distributed computing system. Application code, known as a **job**, executes on an Apache Spark cluster, coordinated by the cluster manager. In general, a job is the highest-level unit of computation. A job represents the complete operation performed by the Spark application. A typical operation includes reading data from a source, applying data transformations, and writing the results to storage or another destination.

Jobs are broken down into **stages**. The job advances through the stages sequentially, which means that later stages must wait for earlier stages to complete. Stages contain groups of identical **tasks** that can be executed in parallel on multiple nodes of the Spark cluster. Tasks are the most granular unit of execution taking place on a subset of the data.

The next sections describe some dashboard visualizations that are useful for performance troubleshooting.

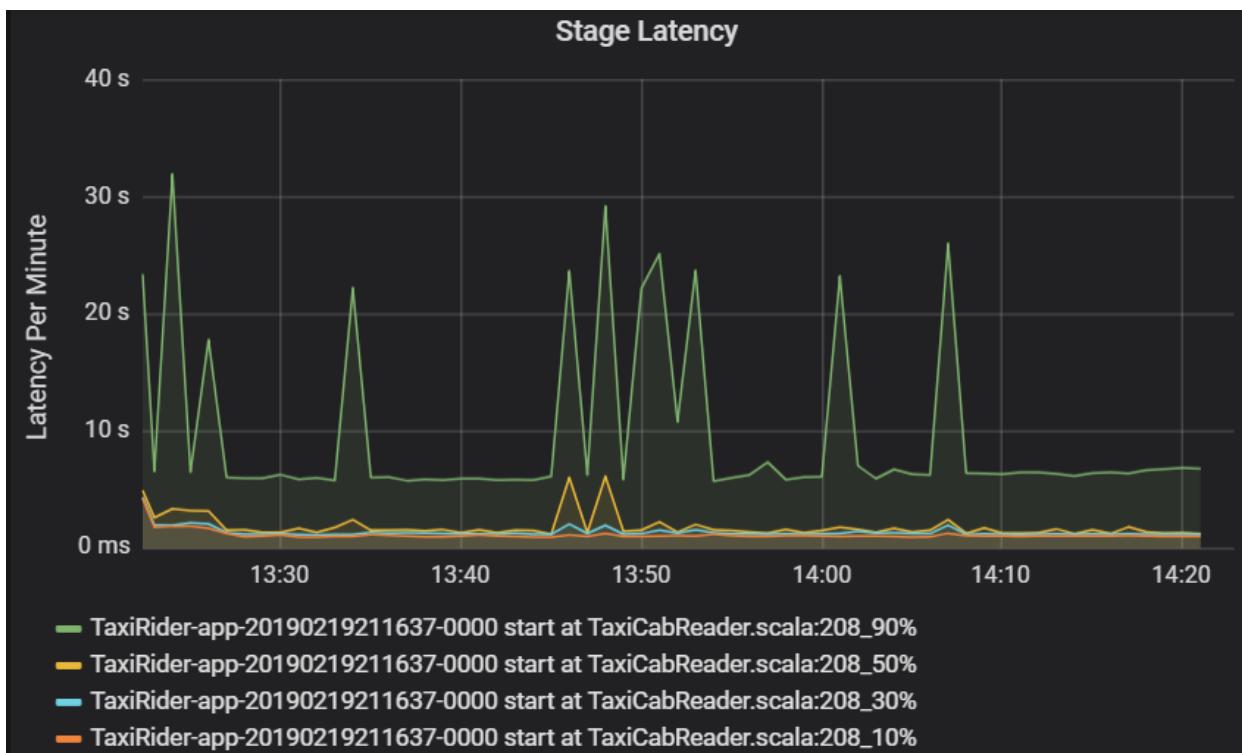
Job and stage latency

Job latency is the duration of a job execution from when it starts until it completes. It is shown as percentiles of a job execution per cluster and application ID, to allow the visualization of outliers. The following graph shows a job history where the 90th percentile reached 50 seconds, even though the 50th percentile was consistently around 10 seconds.

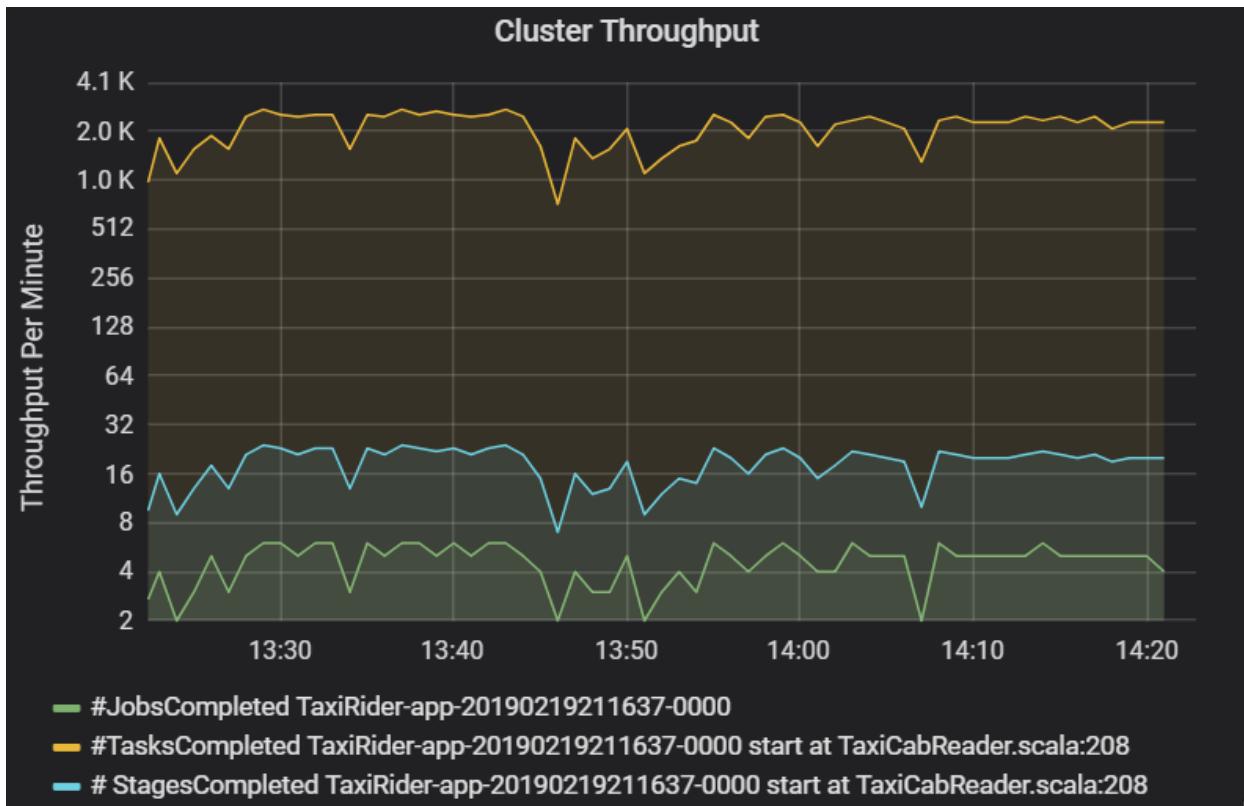


Investigate job execution by cluster and application, looking for spikes in latency. Once clusters and applications with high latency are identified, move on to investigate stage latency.

Stage latency is also shown as percentiles to allow the visualization of outliers. Stage latency is broken out by cluster, application, and stage name. Identify spikes in task latency in the graph to determine which tasks are holding back completion of the stage.

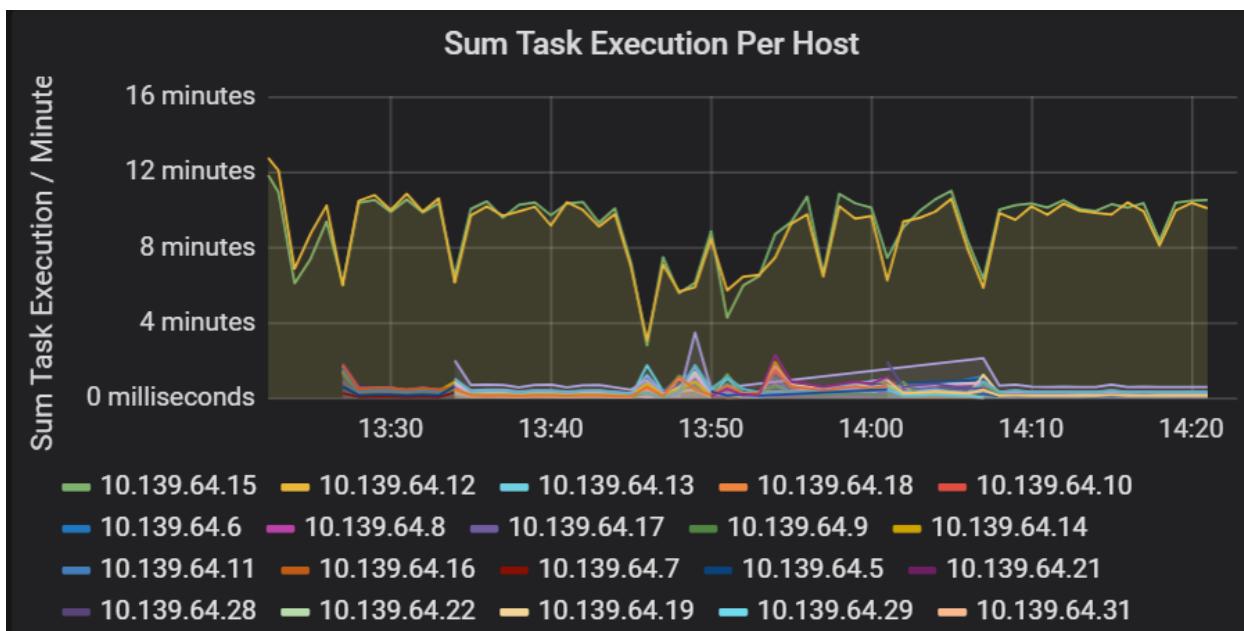


The cluster throughput graph shows the number of jobs, stages, and tasks completed per minute. This helps you to understand the workload in terms of the relative number of stages and tasks per job. Here you can see that the number of jobs per minute ranges between 2 and 6, while the number of stages is about 12 – 24 per minute.

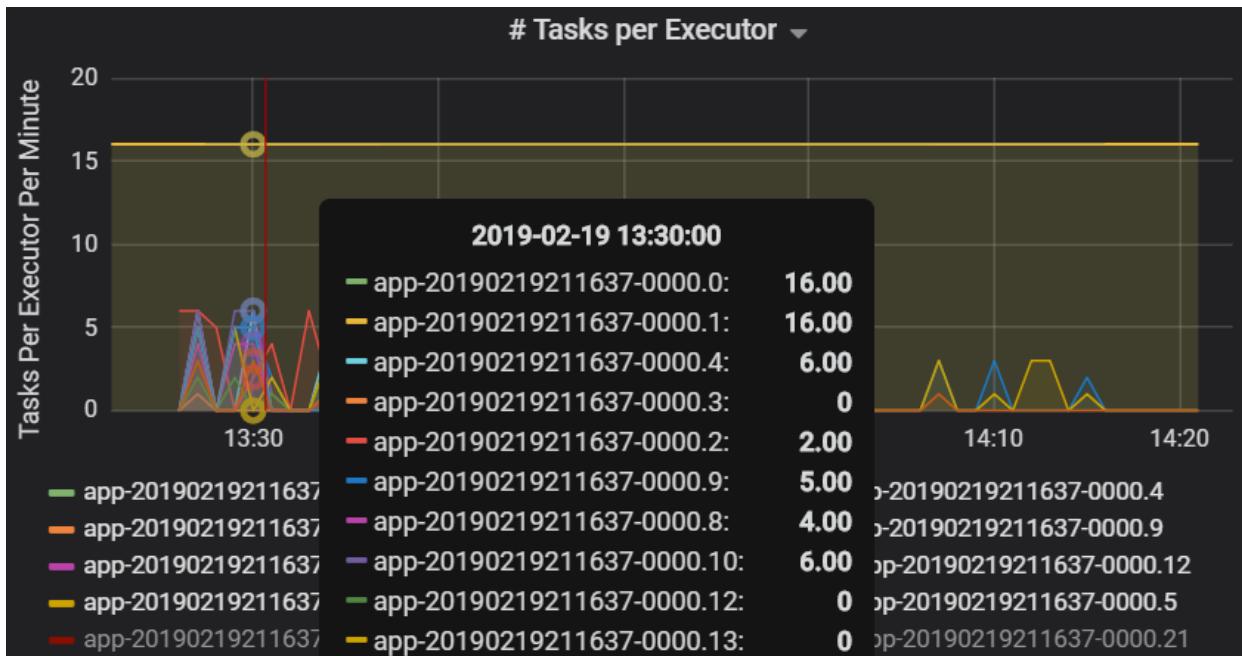


Sum of task execution latency

This visualization shows the sum of task execution latency per host running on a cluster. Use this graph to detect tasks that run slowly due to the host slowing down on a cluster, or a misallocation of tasks per executor. In the following graph, most of the hosts have a sum of about 30 seconds. However, two of the hosts have sums that hover around 10 minutes. Either the hosts are running slow or the number of tasks per executor is misallocated.



The number of tasks per executor shows that two executors are assigned a disproportionate number of tasks, causing a bottleneck.

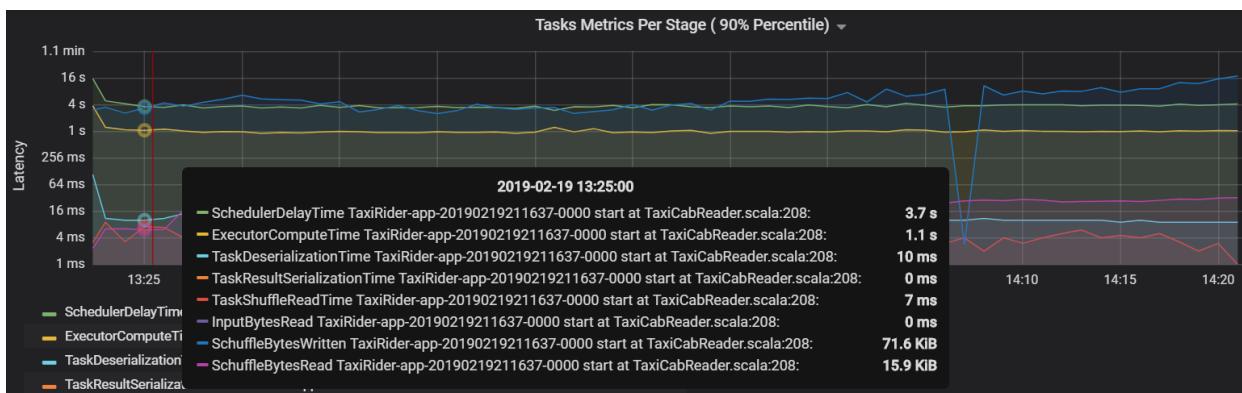


Task metrics per stage

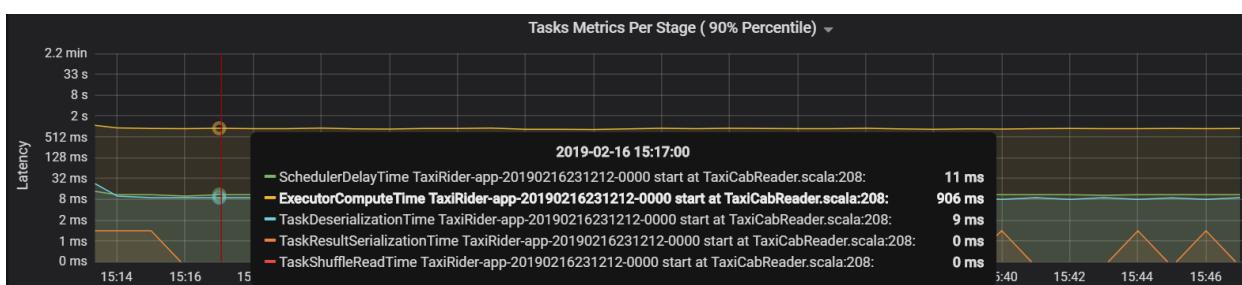
The task metrics visualization gives the cost breakdown for a task execution. You can use it see the relative time spent on tasks such as serialization and deserialization. This data might show opportunities to optimize — for example, by using [broadcast variables](#) to avoid shipping data. The task metrics also show the shuffle data size for a task, and the shuffle read and write times. If these values are high, it means that a lot of data is moving across the network.

Another task metric is the scheduler delay, which measures how long it takes to schedule a task. Ideally, this value should be low compared to the executor compute time, which is the time spent actually executing the task.

The following graph shows a scheduler delay time (3.7 s) that exceeds the executor compute time (1.1 s). That means more time is spent waiting for tasks to be scheduled than doing the actual work.



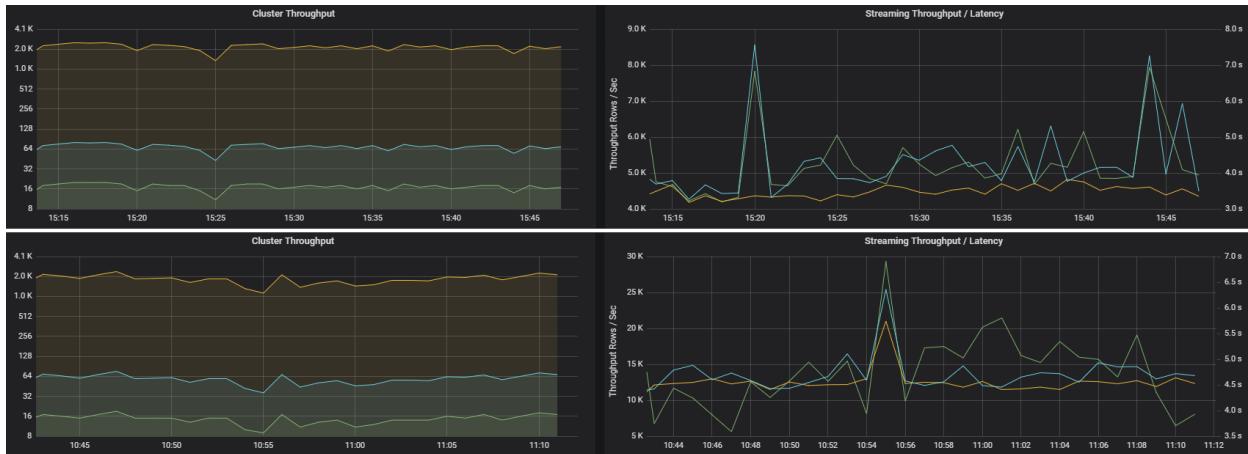
In this case, the problem was caused by having too many partitions, which caused a lot of overhead. Reducing the number of partitions lowered the scheduler delay time. The next graph shows that most of the time is spent executing the task.



Streaming throughput and latency

Streaming throughput is directly related to structured streaming. There are two important metrics associated with streaming throughput: Input rows per second and processed rows per second. If input rows per second outpaces processed rows per second, it means the stream processing system is falling behind. Also, if the input data comes from Event Hubs or Kafka, then input rows per second should keep up with the data ingestion rate at the front end.

Two jobs can have similar cluster throughput but very different streaming metrics. The following screenshot shows two different workloads. They are similar in terms of cluster throughput (jobs, stages, and tasks per minute). But the second run processes 12,000 rows/sec versus 4,000 rows/sec.



Streaming throughput is often a better business metric than cluster throughput, because it measures the number of data records that are processed.

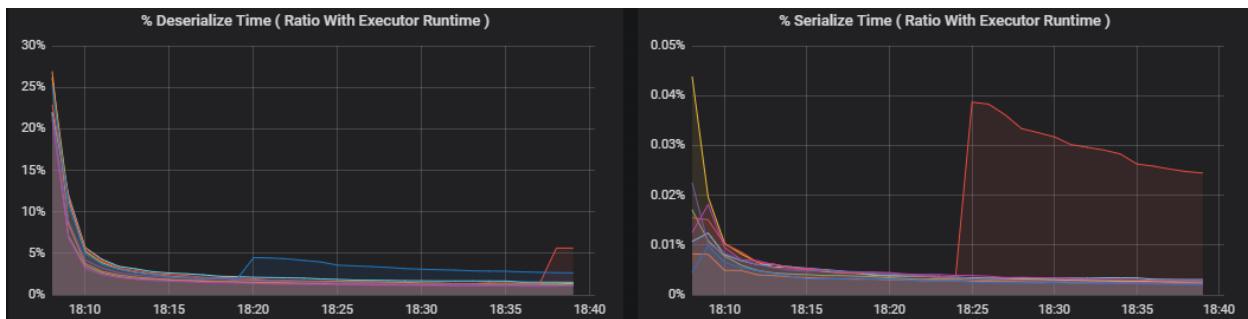
Resource consumption per executor

These metrics help to understand the work that each executor performs.

Percentage metrics measure how much time an executor spends on various things, expressed as a ratio of time spent versus the overall executor compute time. The metrics are:

- % Serialize time
- % Deserialize time
- % CPU executor time
- % JVM time

These visualizations show how much each of these metrics contributes to overall executor processing.



Shuffle metrics are metrics related to data shuffling across the executors.

- Shuffle I/O
- Shuffle memory
- File system usage

- Disk usage

Common performance bottlenecks

Two common performance bottlenecks in Spark are *task stragglers* and a *non-optimal shuffle partition count*.

Task stragglers

The stages in a job are executed sequentially, with earlier stages blocking later stages. If one task executes a shuffle partition more slowly than other tasks, all tasks in the cluster must wait for the slow task to catch up before the stage can end. This can happen for the following reasons:

1. A host or group of hosts are running slow. Symptoms: High task, stage, or job latency and low cluster throughput. The summation of tasks latencies per host won't be evenly distributed. However, resource consumption will be evenly distributed across executors.
2. Tasks have an expensive aggregation to execute (data skewing). Symptoms: High task latency, high stage latency, high job latency, or low cluster throughput, but the summation of latencies per host is evenly distributed. Resource consumption will be evenly distributed across executors.
3. If partitions are of unequal size, a larger partition may cause unbalanced task execution (partition skewing). Symptoms: Executor resource consumption is high compared to other executors running on the cluster. All tasks running on that executor will run slow and hold the stage execution in the pipeline. Those stages are said to be *stage barriers*.

Non-optimal shuffle partition count

During a structured streaming query, the assignment of a task to an executor is a resource-intensive operation for the cluster. If the shuffle data isn't the optimal size, the amount of delay for a task will negatively impact throughput and latency. If there are too few partitions, the cores in the cluster will be underutilized which can result in processing inefficiency. Conversely, if there are too many partitions, there's a great deal of management overhead for a small number of tasks.

Use the resource consumption metrics to troubleshoot partition skewing and misallocation of executors on the cluster. If a partition is skewed, executor resources will be elevated in comparison to other executors running on the cluster.

For example, the following graph shows that the memory used by shuffling on the first two executors is 90X bigger than the other executors:



Run Apache Cassandra on Azure VMs

3/10/2022 • 9 minutes to read • [Edit Online](#)

This article describes performance considerations for running Apache Cassandra on Azure virtual machines.

These recommendations are based on the results of performance tests, which you can find on [GitHub](#). You should use these recommendations as a baseline and then test against your own workload.

Azure Managed Instance for Apache Cassandra

If you're looking for a more automated service for running Apache Cassandra on Azure virtual machines, consider using [Azure Managed Instance for Apache Cassandra](#). This service automates the deployment, management (patching and node health), and scaling of nodes within an Apache Cassandra cluster. It also provides the capability for [hybrid clusters](#), so Apache Cassandra datacenters deployed in Azure can join an existing on-premises or third-party hosted Cassandra ring. The service is deployed by using [Azure virtual machine scale sets](#). The following recommendations were adopted during the development of this service.

Azure VM sizes and disk types

Cassandra workloads on Azure commonly use either [Standard_DS14_v2](#) or [Standard_DS13_v2](#) virtual machines. Cassandra workloads benefit from having more memory in the VM, so consider [memory optimized](#) virtual machine sizes, such as [Standard_DS14_v2](#), or [local-storage optimized](#) sizes such as [Standard_L16s_v2](#).

For durability, data and commit logs are commonly stored on a stripe set of two to four 1-TB [premium managed disks](#) (P30).

Cassandra nodes should not be too data-dense. We recommend having at most 1 – 2 TB of data per VM and enough free space for compaction. To achieve the highest possible combined throughput and IOPS using premium managed disks, we recommend creating a stripe set from a few 1-TB disks, instead of using a single 2-TB or 4-TB disk. For example, on a DS14_v2 VM, four 1-TB disks have a maximum IOPS of $4 \times 5000 = 20\text{ K}$, versus 7.5 K for a single 4-TB disk.

As [Azure Ultra Disks](#) become more widely available across regions, evaluate them for Cassandra workloads that need smaller disk capacity. They can provide higher IOPS/throughput and lower latency on VM sizes like [Standard_D32s_v3](#) and [Standard_D16s_v3](#).

For Cassandra workloads that don't need durable storage — that is, where data can be easily reconstructed from another storage medium — consider using [Standard_L32s_v2](#) or [Standard_L16s_v2](#) VMs. These VMs sizes have large and fast local *temporary* NVMe disks.

For more information, see [Comparing performance of Azure local/ephemeral vs attached/persistent disks](#) (GitHub).

Accelerated Networking

Cassandra nodes make heavy use of the network to send and receive data from the client VM and to communicate between nodes for replication. For optimal performance, Cassandra VMs benefit from high-throughput and low-latency network.

We recommended enabling [Accelerated Networking](#) on the NIC of the Cassandra node and on VMs running client applications accessing Cassandra.

Accelerated networking requires a modern Linux distribution with the latest drivers, such as Cent OS 7.5+ or

Ubuntu 16.x/18.x. For more information, see [Create a Linux virtual machine with Accelerated Networking](#).

Azure VM data disk caching

Cassandra read workloads perform best when random-access disk latency is low. We recommend using Azure managed disks with [ReadOnly](#) caching enabled. ReadOnly caching provides lower average latency, because the data is read from the cache on the host instead of going to the backend storage.

Read-heavy, random-read workloads like Cassandra benefit from the lower read latency even though cached mode has lower throughput limits than uncached mode. (For example, [DS14_v2](#) virtual machines have a maximum cached throughput of 512 MB/s versus uncached of 768 MB/s.)

ReadOnly caching is particularly helpful for Cassandra time-series and other workloads where the working dataset fits in the host cache and data is not constantly overwritten. For example, [DS14_v2](#) provides a cache size of 512 GB, which could store up to 50% of the data from a Cassandra node with 1-2 TB data density.

There is no significant performance penalty from cache-misses when ReadOnly caching is enabled, so cached mode is recommended for all but the most write-heavy workloads.

For more information, see [Comparing Azure VM data disk caching configurations](#) (GitHub).

Linux read-ahead

In most Linux distributions in the Azure Marketplace, the default block device read-ahead setting is 4096 KB. Cassandra's read IOs are usually random and relatively small. Therefore, having a large read-ahead wastes throughput by reading parts of files that aren't needed.

To minimize unnecessary lookahead, set the Linux block device read-ahead setting to 8 KB. (See [Recommended production settings](#) in the DataStax documentation.)

Configure 8 KB read-ahead for all block devices in the stripe set and on the array device itself (for example, `/dev/md0`).

For more information, see [Comparing impact of disk read-ahead settings](#) (GitHub).

Disk array mdadm chunk size

When running Cassandra on Azure, it's common to create an mdadm stripe set (that is, RAID 0) of multiple data disks to increase the overall disk throughput and IOPS closer to the VM limits. Optimal disk stripe size is an application-specific setting. For example, for SQL Server OLTP workloads, the recommendation is 64 KB. For data warehousing workloads, the recommendation is 256 KB.

Our tests found no significant difference between chunk sizes of 64k, 128k, and 256k for Cassandra read workloads. There seems to be a small, slightly noticeable, advantage to the 128k chunk size. Therefore, we recommend the following:

- If you're already using a chunk size of 64 K or 256 K, it doesn't make sense rebuild the disk array to use 128 K size.
- For a new configuration, it makes sense to use 128 K from the beginning.

For more information, see [Measuring impact of mdadm chunk sizes on Cassandra performance](#) (GitHub).

Commit log filesystem

Cassandra writes perform best when commit logs are on disks with high throughput and low latency. In the default configuration, Cassandra 3.x flushes data from memory to the commit log file every ~10 seconds and

doesn't touch the disk for every write. In this configuration, write performance is almost identical whether the commit log is on premium attached disks versus local/ephemeral disks.

Commit logs must be durable, so that a restarted node can reconstruct any data not yet in data files from the flushed commit logs. For better durability, store commit logs on premium managed disks and not on local storage, which can be lost if the VM is migrated to another host.

Based on our tests, Cassandra on CentOS 7.x may have *lower* write performance when commit logs are on the xfs versus ext4 filesystem. Turning on commit log compression brings xfs performance in line with ext4.

Compressed xfs performs as well as compressed and non-compressed ext4 in our tests.

For more information, see [Observations on ext4 and xfs file systems and compressed commit logs](#) (GitHub).

Measuring baseline VM performance

After deploying the VMs for the Cassandra ring, run a few synthetic tests to establish baseline network and disk performance. Use these tests to confirm that performance is in line with expectations, based on the [VM size](#).

Later, when you run the actual workload, knowing the performance baseline makes it easier to investigate potential bottlenecks. For example, knowing the baseline performance for network egress on the VM can help to rule out network as a bottleneck.

For more information about running performance tests, see [Validating baseline Azure VM Performance](#) (GitHub).

Document size

Cassandra read and write performance depends on the document size. You can expect to see higher latency and lower operations/second when reading or writing with larger documents.

For more information, see [Comparing relative performance of various Cassandra document sizes](#) (GitHub).

Replication factor

Most Cassandra workloads use a replication factor (RF) of 3 when using attached premium disks and even 5 when using temporary/ephemeral local disks. The number of nodes in the Cassandra ring should be a multiple of the replication factor. For example, RF 3 implies a ring of 3, 6, 9, or 12 nodes, while RF 5 would have 5, 10, 15, or 20 nodes. When using RF greater than 1 and a consistency level of LOCAL_QUORUM, it's normal for read and write performance to be lower than the same workload running with RF 1.

For more information, see [Comparing relative performance of various replication factors](#) (GitHub).

Linux page caching

When reading data files, Cassandra's Java code uses regular file I/O and benefits from Linux page caching. After parts of the file are read one time, the read content is stored in the OS page cache. Subsequent read access to the same data is much faster.

For this reason, when executing read performance tests against the same data, the second and subsequent reads will appear to be much faster than the original read, which needed to access data on the remote data disk or from the host cache when `ReadOnly` is enabled. To get similar performance measurements on subsequent runs, clear the Linux page cache and restart the Cassandra service to clear its internal memory. When `ReadOnly` caching is enabled, the data might be in the host cache, and subsequent reads will be faster even after clearing the OS page cache and restarting the Cassandra service.

For more information, see [Observations on Cassandra usage of Linux page caching](#) (GitHub).

Multi-datacenter replication

Cassandra natively supports the concept of multiple data centers, making it easy to configure one Cassandra

ring across multiple [Azure regions](#) or across [availability zones](#) within one region.

For a multiregion deployment, use Azure Global VNet-peering to connect the virtual networks in the different regions. When VMs are deployed in the same region but in separate availability zones, the VMs can be in the same virtual network.

It's important to measure the baseline roundtrip latency between regions. Network latency between regions can be 10-100 times higher than latency within a region. Expect a lag between data appearing in the second region when using LOCAL_QUORUM write consistency, or significantly decreased performance of writes when using EACH_QUORUM.

When running Apache Cassandra at scale, and specifically in a multi-DC environment, [node repair](#) becomes challenging. Tools such as [Reaper](#) can help to coordinate repairs at scale (for example, across all the nodes in a data center, one data center at a time, to limit the load on the whole cluster). However, node repair for large clusters is not yet a fully solved problem and applies in all environments, whether on-premises or in the cloud.

When nodes are added to a secondary region, performance will not scale linearly, because some bandwidth and CPU/disk resources are spent on receiving and sending replication traffic across regions.

For more information, see [Measuring impact of multi-dc cross-region replication](#) (GitHub).

Hinted-handoff configuration

In a multiregion Cassandra ring, write workloads with consistency level of LOCAL_QUORUM may lose data in the secondary region. By default, Cassandra hinted handoff is throttled to a relatively low maximum throughput and three-hour hint lifetime. For workloads with heavy writes, we recommended increasing the hinted handoff throttle and hint window time to ensure hints are not dropped before they are replicated.

For more information, see [Observations on hinted handoff in cross-region replication](#) (GitHub).

Next steps

For more information about these performance results, see [Cassandra on Azure VMs Performance Experiments](#).

For information on general Cassandra settings, not specific to Azure, see:

- [DataStax Recommended Production Settings](#)
- [Apache Cassandra Hardware Choices](#)
- [Apache Cassandra Configuration File](#)

The following reference architecture deploys Cassandra as part of an n-tier configuration:

- [Linux N-tier application in Azure with Apache Cassandra](#)

DataOps checklist

3/10/2022 • 3 minutes to read • [Edit Online](#)

DataOps is a lifecycle approach to data analytics. It uses agile practices to orchestrate tools, code, and infrastructure to quickly deliver high-quality data with improved security. When you implement and streamline DataOps processes, your business can more easily and cost effectively deliver analytical insights. This allows you to adopt advanced data techniques that can uncover insights and new opportunities. Use this checklist as a starting point to assess your DataOps process.

Data governance and people

Data governance

- A central location is used to register data sources.
- Data lineage and metadata are available.
- Data is easily discoverable by users, and sensitive data is secured.
- Data and security officers have sightlines into how data is being used, who has access, and where sensitive data might be located.

Defined, clear roles

- Engineers, testers, data scientists, operations, data analysts, business users, and data officers all work together and understand their roles in the project.
- Stakeholders are identified, and you understand what's motivating stakeholders to start making data-driven decisions.

Use cases for data movement

- The use cases for streaming, interactive, and batch analytics are resolved.
- The various types of data for each case are clarified, and metrics are defined to motivate making data-driven decisions.

Data tools

- Data tools needed to make data easier to access, share, analyze, and secure are identified or developed.

Security and compliance

- All resources, data in transit, and data at rest have been audited and meet company security standards.

Development

Pipeline design patterns

- Data pipelines are designed for reuse and use parameterization.
- Pipelines solve common ETL problems.

Centralized ingestion

- A centralized platform hosts pipelines for all external and internal data sources. This allows for simplified management, monitoring, security, and standardization of data movement.
- Costs associated with handling data are also centralized. Central control can help minimize cost and maximize efficiency.

Centralized computations

- A central team defines metrics and determines how to compute those metrics. This allows for consistency across the organization and limits confusion about where to make updates to computations. It also creates one source for metrics definitions, governance, testing, and quality controls.

Data abstraction

- Reporting uses a data abstraction layer. This allows the use of consistent business terminology, a simplified view of data, and minimal effect on data consumers when new versions of the data are made available.

Source control

- Data-related infrastructure, database schemas and procedures, ETL processes, and reports are treated as code and managed in a repository.
- All changes are deployed and tested via a Development, Testing, Acceptance, and Production (DTAP) stack.

Testing and release

DTAP environments

- Non-production environments that mimic the production environment are available.
- Builds and deployments are run and tested on the non-production environment before a production push.
- Developers can deliver reproducible results in all environments.

Testing

- Unit, end-to-end, and regression tests run at a specified frequency and interval.
- All tests are in source control and run as part of a build and deploy process.
- Post-deployment end-user input is welcome and incorporated into testing as appropriate.

Build and deploy process

- A gated process deploys changes to the production environment.
- Changes are tested in the development and test environments. Changes are certified before they go to production. This process is as automated as possible.

Monitoring

Alerting and remediation

- Operations is alerted to any errors.
- You can respond to feedback quickly and have a process for quickly addressing issues as they arise.
- Pipelines are observable.

Efficiency

- Data movement is efficient.
- Infrastructure can be scaled to meet volume and velocity needs.
- Data is reusable whenever possible.

Statistical process control (SPC)

- SPC is used to monitor and control the data pipelines.
- You can use the outputs of pipelines to determine the next step in the data flow.

Next steps

- Organize data operations team members
- DevOps automation for data management and analytics in Azure
- Smart data pipelines to Azure: Ingesting and migrating data the DataOps way

Related resources

- [DataOps for the modern data warehouse](#)
- [Team Data Science Process for DevOps](#)

Choose a data analytics technology in Azure

3/10/2022 • 4 minutes to read • [Edit Online](#)

The goal of most big data solutions is to provide insights into the data through analysis and reporting. This can include preconfigured reports and visualizations, or interactive data exploration.

What are your options when choosing a data analytics technology?

There are several options for analysis, visualizations, and reporting in Azure, depending on your needs:

- [Power BI](#)
- [Jupyter Notebooks](#)
- [Zeppelin Notebooks](#)
- [Microsoft Azure Notebooks](#)

Power BI

[Power BI](#) is a suite of business analytics tools. It can connect to hundreds of data sources, and can be used for ad hoc analysis. See [this list](#) of the currently available data sources. Use [Power BI Embedded](#) to integrate Power BI within your own applications without requiring any additional licensing.

Organizations can use Power BI to produce reports and publish them to the organization. Everyone can create personalized dashboards, with governance and [security built in](#). Power BI uses [Azure Active Directory](#) (Azure AD) to authenticate users who log in to the Power BI service, and uses the Power BI login credentials whenever a user attempts to access resources that require authentication.

Jupyter Notebooks

[Jupyter Notebooks](#) provide a browser-based shell that lets data scientists create *notebook* files that contain Python, Scala, or R code and markdown text, making it an effective way to collaborate by sharing and documenting code and results in a single document.

Most varieties of HDInsight clusters, such as Spark or Hadoop, come [preconfigured with Jupyter notebooks](#) for interacting with data and submitting jobs for processing. Depending on the type of HDInsight cluster you are using, one or more kernels will be provided for interpreting and running your code. For example, Spark clusters on HDInsight provide Spark-related kernels that you can select from to execute Python or Scala code using the Spark engine.

Jupyter notebooks provide a great environment for analyzing, visualizing, and processing your data prior to building more advanced visualizations with a BI/reporting tool like Power BI.

Zeppelin Notebooks

[Zeppelin Notebooks](#) are another option for a browser-based shell, similar to Jupyter in functionality. Some HDInsight clusters come [preconfigured with Zeppelin notebooks](#). However, if you are using an [HDInsight Interactive Query](#) (Hive LLAP) cluster, [Zeppelin](#) is currently your only choice of notebook that you can use to run interactive Hive queries. Also, if you are using a [domain-joined HDInsight cluster](#), Zeppelin notebooks are the only type that enables you to assign different user logins to control access to notebooks and the underlying Hive tables.

Microsoft Azure Notebooks

[Azure Notebooks](#) is an online Jupyter Notebooks-based service that enables data scientists to create, run, and share Jupyter Notebooks in cloud-based libraries. Azure Notebooks provides execution environments for Python 2, Python 3, F#, and R, and provides several charting libraries for visualizing your data, such as ggplot,

matplotlib, bokeh, and seaborn.

Unlike Jupyter notebooks running on an HDInsight cluster, which are connected to the cluster's default storage account, Azure Notebooks does not provide any data. You must [load data](#) in a variety of ways, such downloading data from an online source, interacting with Azure Blobs or Table Storage, connecting to a SQL database, or loading data with the Copy Wizard for Azure Data Factory.

Key benefits:

- Free service—no Azure subscription required.
- No need to install Jupyter and the supporting R or Python distributions locally—just use a browser.
- Manage your own online libraries and access them from any device.
- Share your notebooks with collaborators.

Considerations:

- You will be unable to access your notebooks when offline.
- Limited processing capabilities of the free notebook service may not be enough to train large or complex models.

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need to connect to numerous data sources, providing a centralized place to create reports for data spread throughout your domain? If so, choose an option that allows you to connect to 100s of data sources.
- Do you want to embed dynamic visualizations in an external website or application? If so, choose an option that provides embedding capabilities.
- Do you want to design your visualizations and reports while offline? If yes, choose an option with offline capabilities.
- Do you need heavy processing power to train large or complex AI models or work with very large data sets? If yes, choose an option that can connect to a big data cluster.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Connect to big data cluster for advanced processing	Yes	Yes	Yes	No
Managed service	Yes	Yes ¹	Yes ¹	Yes
Connect to 100s of data sources	Yes	No	No	No
Offline capabilities	Yes ²	No	No	No

CAPABILITY	POWER BI	JUPYTER NOTEBOOKS	ZEPELIN NOTEBOOKS	MICROSOFT AZURE NOTEBOOKS
Embedding capabilities	Yes	No	No	No
Automatic data refresh	Yes	No	No	No
Access to numerous open source packages	No	Yes ³	Yes ³	Yes ⁴
Data transformation/cleaning options	Power Query, R	40 languages, including Python, R, Julia, and Scala	20+ interpreters, including Python, JDBC, and R	Python, F#, R
Pricing	Free for Power BI Desktop (authoring), see pricing for hosting options	Free	Free	Free
Multiuser collaboration	Yes	Yes (through sharing or with a multiuser server like JupyterHub)	Yes	Yes (through sharing)

[1] When used as part of a managed HDInsight cluster.

[2] With the use of Power BI Desktop.

[2] You can search the [Maven repository](#) for community-contributed packages.

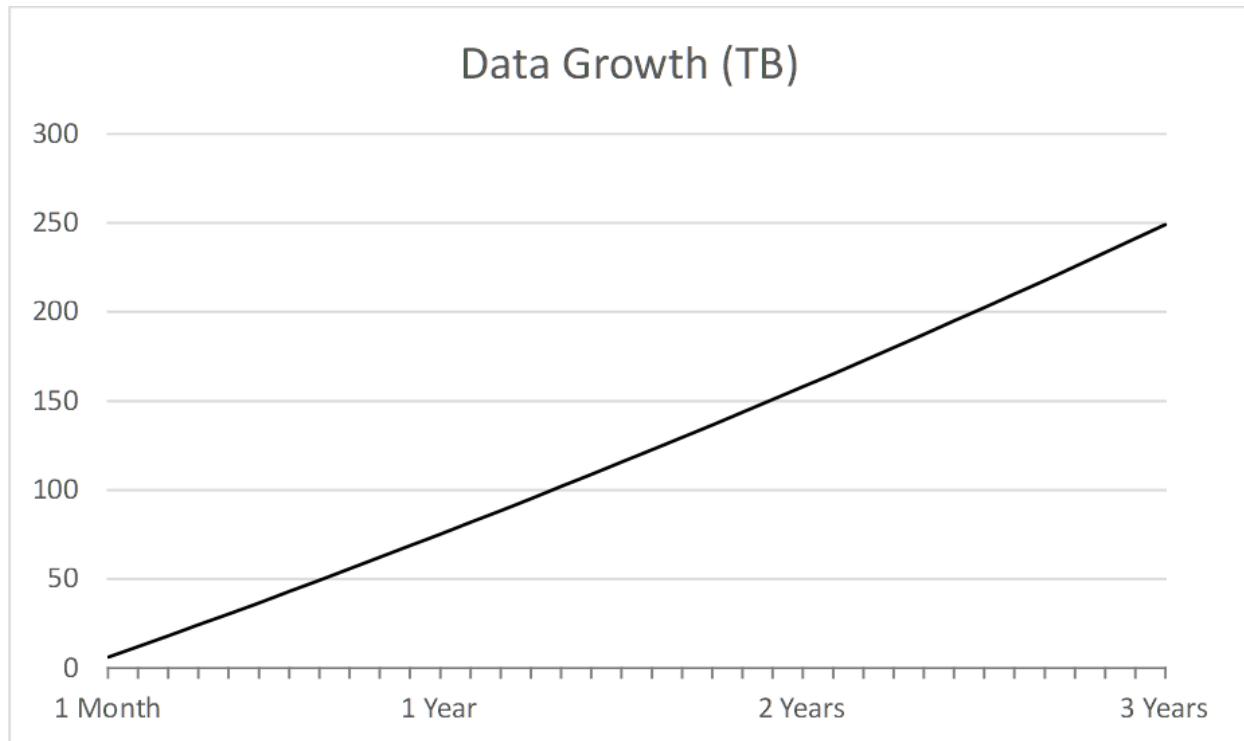
[3] Python packages can be installed using either pip or conda. R packages can be installed from CRAN or GitHub. Packages in F# can be installed via nuget.org using the [Paket dependency manager](#).

Build a scalable system for massive data

3/10/2022 • 2 minutes to read • [Edit Online](#)

Your data storage system is fundamental to the success of your applications, and therefore to the success of your enterprise. When the storage system is well architected, response is quick, data storage capacity is easily adjusted as necessary, the system is resilient to failures, and it's affordable.

A crucial consideration is whether the design scales well as data grows. As an example of data growth, consider an application that generates 6 terabytes (TB) of data its first month, with the amount increasing every month at a 10 percent yearly rate. Here's a graph that shows how data accumulates over time:



After three years, there's 249 TB of data. If the system is well architected, it handles such data growth gracefully, remaining responsive, resilient, and affordable.

This example isn't extreme. If your customers are businesses, data grows both as you add customers and as your customers add data. It can also grow because of application enhancements.

Handling data growth may require a mix of storage products. For example, you may need to keep rarely accessed data in low-cost services, and frequently accessed data in higher-cost services with better access times.

To design such a system on Azure, you need to be familiar with the many Azure services and with how to use them for various types of applications and various objectives. The articles in this section provide seven system architectures for web applications that use massive amounts of data and that are resilient to system failures. They serve as examples that can help you design a storage system that properly accommodates your applications.

The architectures demonstrate the use of these Azure products: Azure Table Storage, Azure Cosmos DB, Azure Data Factory, and Azure Data Lake.

This capability matrix provides links to the articles and summarizes the benefits and risks of each architecture:

ARCHITECTURE	BENEFITS	RISKS
Two-region web application with Table Storage failover	Straightforward, low-cost implementation	Limited resiliency—only two Azure regions
Multi-region web application with custom Storage Table replication	Resiliency	Implementation time and difficulty
Multi-region web application with Cosmos DB replication	Resiliency, performance, scalability	Storage costs
Optimized storage with logical data classification	Resiliency, performance, scalability, storage costs	Implementation time, need to design logical data classification
Optimized Storage – time based – multi writes	Storage costs	Limited resiliency, performance, limited scalability, implementation time, need to design time-based data retention
Optimized Storage – time based with Data Lake	Resiliency, performance, scalability	Implementation time, need to design time-based data retention
Minimal storage – change feed to replicate data	Resiliency, performance, time-based data retention	Limited scalability, implementation time

Next steps

Here are resources to help you design your storage solution and investigate its business aspects, including costs and service-level agreements.

Design storage solutions

- [Build great solutions with the Microsoft Azure Well-Architected Framework](#)
- [Understand data store models](#)
- [Select an Azure data store for your application](#)
- [Criteria for choosing a data store](#)
- [Choose a data storage approach in Azure](#)
- [Developing with Azure Cosmos DB Table API and Azure Table storage](#)

Azure service limits, cost, service level agreements (SLA), and regional availability

- [Azure subscription and service limits, quotas, and constraints](#)
- [Azure pricing](#)
- [Service-level agreements](#)
- [Products available by region](#)

Microservices assessment and readiness

3/10/2022 • 16 minutes to read • [Edit Online](#)

A microservices architecture can provide many benefits for your applications, including agility, scalability, and high availability. Along with these benefits, this architecture presents challenges. When you build microservices-based applications or transform existing applications into a microservices architecture, you need to analyze and assess your current situation to identify areas that need improvement.

This guide will help you understand some considerations to keep in mind when you move to a microservices architecture. You can use this guide to assess the maturity of your application, infrastructure, DevOps, development model, and more.

Understand business priorities

To start evaluating a microservices architecture, you need to first understand the core priorities of your business. Core priorities might be related to agility, change adoption, or rapid development, for example. You need to analyze whether your architecture is a good fit for your core priorities. Keep in mind that business priorities can change over time. For example, innovation is a top priority for startups, but after a few years, the core priorities might be reliability and efficiency.

Here are some priorities to consider:

- Innovation
- Reliability
- Efficiency

Document the SLAs that are aligned with various parts of your application to ensure an organizational commitment that can serve as a guide to your assessment.

Record architectural decisions

A microservices architecture helps teams become autonomous. Teams can make their own decisions about technologies, methodologies, and infrastructure components, for example. However, these choices should respect the formally agreed-upon principles known as shared governance, which express the agreement among teams on how to address the broader strategy for microservices.

Consider these factors:

- Is shared governance in place?
- Do you track decisions and their trade-offs in an architecture journal?
- Can your team easily access your architecture journal?
- Do you have a process for evaluating tools, technologies, and frameworks?

Assess team composition

You need to have the proper team structure to avoid unnecessary communication across teams. A microservices architecture encourages the formation of small, focused, cross-functional teams and requires a mindset change, which must be preceded by team restructuring.

Consider these factors:

- Are your teams split based on subdomains, following [domain-driven design \(DDD\) principles](#)?

- Are your teams cross-functional, with enough capacity to build and operate related microservices independently?
- How much time is spent in ad hoc activities and tasks that aren't related to projects?
- How much time is spent in cross-team collaboration?
- Do you have a process for identifying and minimizing technical debt?
- How are lessons learned and experience communicated across teams?

Use the Twelve-Factor methodology

The fundamental goal of choosing a microservices architecture is to deliver value faster and be adaptive to change by following agile practices. The [Twelve-Factor app methodology](#) provides guidelines for building maintainable and scalable applications. These guidelines promote attributes like immutability, ephemerality, declarative configuration, and automation. By incorporating these guidelines and avoiding common pitfalls, you can create loosely coupled, self-contained microservices.

Understand the decomposition approach

Transforming a monolithic application to a microservices architecture takes time. Start with edge services. Edge services have fewer dependencies on other services and can be easily decomposed from the system as independent services. We highly recommend patterns like [Strangler Fig](#) and [Anti-corruption Layer](#) to keep the monolithic application in a working state until all services are decomposed into separate microservices. During segregation, the principles of DDD can help teams choose components or services from the monolithic application based on subdomains.

For example, in an e-commerce system, you might have these modules: cart, product management, order management, pricing, invoice generation, and notification. You decide to start the transformation of the application with the notification module because it doesn't have dependencies on other modules. However, other modules might depend on this module to send out notifications. The notification module can easily be decomposed into a separate microservice, but you'll need to make some changes in the monolithic application to call the new notification service. You decide to transform the invoice generation module next. This module is called after an order is generated. You can use patterns like Strangler and Anti-corruption to support this transformation.

Data synchronization, multi-writes to both monolithic and microservice interfaces, data ownership, schema decomposition, joins, volume of data, and data integrity might make data breakdown and migration difficult. There are several techniques that you can use, like keeping a shared database between microservices, decoupling databases from a group of services based on business capability or domain, and isolating databases from the services. The recommended solution is to decompose each database with each service. In many circumstances, that's not practical. In those cases, you can use patterns like the Database View pattern and the Database Wrapping Service pattern.

Assess DevOps readiness

When you move to a microservices architecture, it's important to assess your DevOps competence. A microservices architecture is intended to facilitate agile development in applications to increase organizational agility. DevOps is one of the key practices that you should implement to achieve this competence.

When you evaluate your DevOps capability for a microservices architecture, keep these points in mind:

- Do people in your organization know the fundamental practices and principles of DevOps?
- Do teams understand source control tools and their integration with CI/CD pipelines?
- Do you implement DevOps practices properly?
 - Do you follow agile practices?

- Is continuous integration implemented?
- Is continuous delivery implemented?
- Is continuous deployment implemented?
- Is continuous monitoring implemented?
- Is [Infrastructure as Code \(IaC\)](#) in place?
- Do you use the right tools to support CI/CD?
- How is configuration of staging and production environments managed for the application?
- Does the tool chain have community support and a support model and provide proper channels and documentation?

Identify business areas that change frequently

A microservices architecture is flexible and adaptable. During assessment, drive a discussion in the organization to determine the areas that your colleagues think will change frequently. Building microservices allows teams to quickly respond to changes that customers request and minimize regression testing efforts. In a monolithic application, a change in one module requires numerous levels of regression testing and reduces agility in releasing new versions.

Consider these factors:

- Is the service independently deployable?
- Does the service follow DDD principles?
- Does the service follow [SOLID](#) principles?
- Is the database private to the service?
- Did you build the service by using the supported microservices chassis pattern?

Assess infrastructure readiness

When you shift to a microservices architecture, infrastructure readiness is a critical point to consider. The application's performance, availability, and scalability will be affected if the infrastructure isn't properly set up or if the right services or components aren't used. Sometimes an application is created with all the suggested methodologies and procedures, but the infrastructure is inadequate. This results in poor performance and maintenance.

Consider these factors when you evaluate your infrastructure readiness:

- Does the infrastructure ensure the scalability of the services deployed?
- Does the infrastructure support provisioning through scripts that can be automated via CI/CD?
- Does the deployment infrastructure offer an SLA for availability?
- Do you have a disaster recovery (DR) plan and routine drill schedules?
- Is the data replicated to different regions for DR?
- Do you have a data backup plan?
- Are the deployment options documented?
- Is the deployment infrastructure monitored?
- Does the deployment infrastructure support self-healing of services?

Assess release cycles

Microservices are adaptive to change and take advantage of agile development to shorten release cycles and bring value to customers more. Consider these factors when you evaluate your release cycles:

- How often do you build and release applications?

- How often do your releases fail after deployment?
- How long does it take to recover or remediate problems after an outage?
- Do you use semantic versioning for your applications?
- Do you maintain different environments and propagate the same release in a sequence (for example, first to staging and then to production)?
- Do you use versioning for your APIs?
- Do you follow proper versioning guidelines for APIs?
- When do you change an API version?
- What's your approach for handling API versioning?
 - URI path versioning
 - Query parameter versioning
 - Content-type versioning
 - Custom header versioning
- Do you have a practice in place for event versioning?

Assess communication across services

Microservices are self-contained services that communicate with each other across process boundaries to address business scenarios. To get reliable and dependable communication, you need to select an appropriate communication protocol.

Take these factors into consideration:

- Are you following an API-first approach, where APIs are treated as first-class citizens?
- Do you have long-chain operations, where multiple services communicate in sequence over a synchronous communication protocol?
- Have you considered asynchronous communication anywhere in the system?
- Have you assessed the message broker technology and its capabilities?
- Do you understand the throughput of messages that services receive and process?
- Do you use direct client-to-service communication?
- Do you need to persist messages at the message broker level?
- Are you using the [Materialized View pattern](#) to address the chatty behavior of microservices?
- Have you implemented Retry, Circuit Breaker, Exponential Backoff, and Jitter for reliable communication? A common way to handle this is to use the [Ambassador pattern](#).
- Do you have defined domain events to facilitate communication between microservices?

Evaluate how services are exposed to clients

An API gateway is one of the core components in a microservices architecture. Directly exposing services to the clients creates problems in terms of manageability, control, and dependable communication. An API gateway serves as a proxy server, intercepting traffic and routing it to back-end services. If you need to filter traffic by IP address, authorization, mock responses, and so on, you can do it at the API gateway level without making any changes to the services.

Take these factors into consideration:

- Are the services directly consumed by clients?
- Is there an API gateway that acts as a facade for all the services?
- Can the API gateway set up policies like quota limits, mocking responses, and filtering of IP addresses?
- Are you using multiple API gateways to address the needs of various types of clients, like mobile apps, web apps, and partners?

- Does your API gateway provide a portal where clients can discover and subscribe to services, like a developer portal in [Azure API Management](#)?
- Does your solution provide L7 load balancing or Web Application Firewall (WAF) capabilities along with the API gateway?

Assess transaction handling

Distributed transactions facilitate the execution of multiple operations as a single unit of work. In a microservices architecture, the system is decomposed into numerous services. A single business use case is addressed by multiple microservices as part of a single distributed transaction. In a distributed transaction, a command is an operation that starts when an event occurs. The event triggers the an operation in the system. If the operation succeeds, it might trigger another command, which can then trigger another event, and so on until all the transactions are completed or rolled back, depending on whether the transaction succeeds.

Take the following considerations into account:

- How many distributed transactions are there in the system?
- What's your approach to handling distributed transactions? Evaluate the use of the [Saga pattern](#) with orchestration or choreography.
- How many transactions span multiple services?
- Are you following an ACID or BASE transaction model to achieve data consistency and integrity?
- Are you using long-chaining operations for transactions that span multiple services?

Assess your service development model

One of the greatest benefits of microservices architectures is technology diversity. Microservices-based systems enable teams to develop services by using the technologies that they choose. In traditional application development, you might reuse existing code when you build new components, or create an internal development framework to reduce development effort. The use of multiple technologies can prevent code reuse.

Consider these factors:

- Do you use a service template to kickstart new service development?
- Do you follow the Twelve-Factor app methodology and use a single code base for microservices, isolating dependencies and externalizing configuration?
- Do you keep sensitive configuration in key vaults?
- Do you containerize your services?
- Do you know your data consistency requirements?

Assess your deployment approach

Your deployment approach is your method for releasing versions of your application across various deployment environments. Microservices-based systems provide the agility to release versions faster than you can with traditional applications.

When you assess your deployment plan, consider these factors:

- Do you have a strategy for deploying your services?
- Are you using modern tools and technologies to deploy your services?
- What kind of collaboration is required among teams when you deploy services?
- Do you provision infrastructure by using Infrastructure as Code (IaC)?
- Do you use DevOps capabilities to automate deployments?

- Do you propagate the same builds to multiple environments, as suggested by the Twelve-Factor app methodology?

Assess your hosting platform

Scalability is one of the key advantages of microservices architectures. That's because microservices are mapped to business domains, so each service can be scaled independently. A monolithic application is deployed as a single unit on a hosting platform and needs to be scaled holistically. That results in downtime, deployment risk, and maintenance. Your monolithic application might be well designed, with components addressing individual business domains. But because of a lack of process boundaries, the potential for violating the principles of single responsibility becomes more difficult. This eventually results in spaghetti code. Because the application is composed and deployed as a single hosting process, scalability is difficult.

Microservices enables teams to choose the right hosting platform to support their scalability needs. Various hosting platforms are available to address these challenges by providing capabilities like autoscaling, elastic provisioning, higher availability, faster deployment, and easy monitoring.

Consider these factors:

- What hosting platform do you use to deploy your services (virtual machines, containers, serverless)?
- Is the hosting platform scalable? Does it support autoscaling?
- How long does it take to scale your hosting platform?
- Do you understand the SLAs that various hosting platforms provide?
- Does your hosting platform support disaster recovery?

Assess services monitoring

Monitoring is an important element of a microservices-based application. Because the application is divided into a number of services that run independently, troubleshooting errors can be daunting. If you use proper semantics to monitor your services, your teams can easily monitor, investigate, and respond to errors.

Consider these factors:

- Do you monitor your deployed services?
- Do you have a logging mechanism? What tools do you use?
- Do you have a distributed tracing infrastructure in place?
- Do you record exceptions?
- Do you maintain business error codes for quick identification of problems?
- Do you use health probes for services?
- Do you use semantic logging? Have you implemented key metrics, thresholds, and indicators?
- Do you mask sensitive data during logging?

Assess correlation token assignment

In a microservices architecture, an application is composed of various microservices that are hosted independently, interacting with each other to serve various business use cases. When an application interacts with back-end services to perform an operation, you can assign a unique number, known as a correlation token, to the request. We recommend that you consider using correlation tokens, because they can help you troubleshoot errors. They help you determine the root cause of a problem, assess the impact, and determine an approach to remediate the problem.

You can use correlation tokens to retrieve the request trail by identifying which services contain the correlation token and which don't. The services that don't have the correlation token for that request failed. If a failure occurs, you can later retry the transaction. To enforce idempotency, only services that don't have the correlation

token will serve the request.

For example, if you have a long chain of operations that involves many services, passing a correlation token to services can help you investigate issues easily if any of the services fails during a transaction. Each service usually has its own database. The correlation token is kept in the database record. In case of a transaction replay, services that have that particular correlation token in their databases ignore the request. Only services that don't have the token serve the request.

Consider these factors:

- At which stage do you assign the correlation token?
- Which component assigns the correlation token?
- Do you save correlation tokens in the service's database?
- What's the format of correlation tokens?
- Do you log correlation tokens and other request information?

Evaluate the need for a microservices chassis framework

A microservices chassis framework is a base framework that provides capabilities for cross-cutting concerns like logging, exception handling, distributed tracing, security, and communication. When you use a chassis framework, you focus more on implementing the service boundary than interacting with infrastructure functionality.

For example, say you're building a cart management service. You want to validate the incoming token, write logs to the logging database, and communicate with another service by invoking that service's endpoint. If you use a microservices chassis framework, you can reduce development efforts. Dapr is one system that provides various building blocks for implementing cross-cutting concerns.

Consider these factors:

- Do you use a microservices chassis framework?
- Do you use Dapr to interact with cross-cutting concerns?
- Is your chassis framework language agnostic?
- Is your chassis framework generic, so it supports all kinds of applications? A chassis framework shouldn't contain application-specific logic.
- Does your chassis framework provide a mechanism to use the selected components or services as needed?

Assess your approach to application testing

Traditionally, testing is done after development is completed and the application is ready to roll out to user acceptance testing (UAT) and production environments. There's currently a shift in this approach, moving the testing early (left) in the application development lifecycle. Shift-left testing increases the quality of applications because testing is done during each phase of the application development lifecycle, including the design, development, and post-development phases.

For example, when you build an application, you start by designing an architecture. When you use the shift-left approach, you test the design for vulnerabilities by using tools like [Microsoft Threat Modeling](#). When you start development, you can scan your source code by running tools like static application security testing (SAST) and using other analyzers to uncover problems. After you deploy the application, you can use tools like dynamic application security testing (DAST) to test it while it's hosted. Functional testing, chaos testing, penetration testing, and other kinds of testing happen later.

Consider these factors:

- Do you write test plans that cover the entire development lifecycle?

- Do you include testers in requirements meetings and in the entire application development lifecycle?
- Do you use test-driven design or behavior-driven design?
- Do you test user stories? Do you include acceptance criteria in your user stories?
- Do you test your design by using tools like Microsoft Threat Modeling?
- Do you write unit tests?
- Do you use static code analyzers or other tools to measure code quality?
- Do you use automated tools to test applications?
- Do you implement [Secure DevOps](#) practices?
- Do you do integration testing, end-to-end application testing, load/performance testing, penetration testing, and chaos testing?

Assess microservices security

Service protection, secure access, and secure communication are among the most important considerations for a microservices architecture. For example, a microservices-based system that spans multiple services and uses token validation for each service isn't a viable solution. This system would affect the agility of the overall system and the potential of introducing implementation drift across services.

Security concerns are usually handled by the API gateway and the application firewall. The gateway and firewall take incoming requests, validate tokens, and apply various filters and policies, like implementing OWASP Top 10 principles to intercept traffic. Finally, they send the request to the back-end microservices. This configuration helps developers focus on business needs rather than the cross-cutting concern of security.

Consider these factors:

- Are authentication and authorization required for the service?
- Are you using an API gateway to validate tokens and incoming requests?
- Are you using SSL or mTLS to provide security for service communication?
- Do you have network security policies in place to allow the required communication among services?
- Are you using firewalls (L4, L7) where applicable to provide security for internal and external communications?
- Do you use security policies in API Gateway to control traffic?

Next steps

- [Microservices on Azure](#)
- [Embrace Microservices Design](#)
- [Introduction to deployment patterns](#)
- [Design a microservices-oriented application](#)

Related resources

- [Microservices architecture style](#)
- [Build microservices on Azure](#)
- [Microservices architecture on Azure Kubernetes Service](#)
- [Using domain analysis to model microservices](#)
- [Using tactical DDD to design microservices](#)
- [Design a microservices architecture](#)

Using domain analysis to model microservices

3/10/2022 • 8 minutes to read • [Edit Online](#)

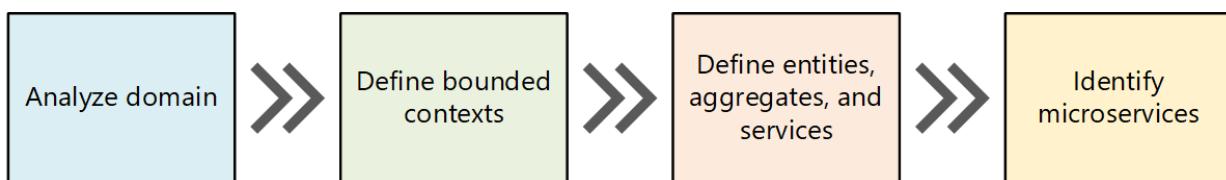
One of the biggest challenges of microservices is to define the boundaries of individual services. The general rule is that a service should do "one thing" — but putting that rule into practice requires careful thought. There is no mechanical process that will produce the "right" design. You have to think deeply about your business domain, requirements, and goals. Otherwise, you can end up with a haphazard design that exhibits some undesirable characteristics, such as hidden dependencies between services, tight coupling, or poorly designed interfaces. This article shows a domain-driven approach to designing microservices.

This article uses a drone delivery service as a running example. You can read more about the scenario and the corresponding reference implementation [here](#).

Introduction

Microservices should be designed around business capabilities, not horizontal layers such as data access or messaging. In addition, they should have loose coupling and high functional cohesion. Microservices are *loosely coupled* if you can change one service without requiring other services to be updated at the same time. A microservice is *cohesive* if it has a single, well-defined purpose, such as managing user accounts or tracking delivery history. A service should encapsulate domain knowledge and abstract that knowledge from clients. For example, a client should be able to schedule a drone without knowing the details of the scheduling algorithm or how the drone fleet is managed.

Domain-driven design (DDD) provides a framework that can get you most of the way to a set of well-designed microservices. DDD has two distinct phases, strategic and tactical. In strategic DDD, you are defining the large-scale structure of the system. Strategic DDD helps to ensure that your architecture remains focused on business capabilities. Tactical DDD provides a set of design patterns that you can use to create the domain model. These patterns include entities, aggregates, and domain services. These tactical patterns will help you to design microservices that are both loosely coupled and cohesive.



In this article and the next, we'll walk through the following steps, applying them to the Drone Delivery application:

1. Start by analyzing the business domain to understand the application's functional requirements. The output of this step is an informal description of the domain, which can be refined into a more formal set of domain models.
2. Next, define the *bounded contexts* of the domain. Each bounded context contains a domain model that represents a particular subdomain of the larger application.
3. Within a bounded context, apply tactical DDD patterns to define entities, aggregates, and domain services.
4. Use the results from the previous step to identify the microservices in your application.

In this article, we cover the first three steps, which are primarily concerned with DDD. In the next article, we'll identify the microservices. However, it's important to remember that DDD is an iterative, ongoing process.

Service boundaries aren't fixed in stone. As an application evolves, you may decide to break apart a service into several smaller services.

NOTE

This article doesn't show a complete and comprehensive domain analysis. We deliberately kept the example brief, to illustrate the main points. For more background on DDD, we recommend Eric Evans' *Domain-Driven Design*, the book that first introduced the term. Another good reference is *Implementing Domain-Driven Design* by Vaughn Vernon.

Scenario: Drone delivery

Fabrikam, Inc. is starting a drone delivery service. The company manages a fleet of drone aircraft. Businesses register with the service, and users can request a drone to pick up goods for delivery. When a customer schedules a pickup, a backend system assigns a drone and notifies the user with an estimated delivery time. While the delivery is in progress, the customer can track the location of the drone, with a continuously updated ETA.

This scenario involves a fairly complicated domain. Some of the business concerns include scheduling drones, tracking packages, managing user accounts, and storing and analyzing historical data. Moreover, Fabrikam wants to get to market quickly and then iterate quickly, adding new functionality and capabilities. The application needs to operate at cloud scale, with a high service level objective (SLO). Fabrikam also expects that different parts of the system will have very different requirements for data storage and querying. All of these considerations lead Fabrikam to choose a microservices architecture for the Drone Delivery application.

Analyze the domain

Using a DDD approach will help you to design microservices so that every service forms a natural fit to a functional business requirement. It can help you to avoid the trap of letting organizational boundaries or technology choices dictate your design.

Before writing any code, you need a bird's eye view of the system that you are creating. DDD starts by modeling the business domain and creating a *domain model*. The domain model is an abstract model of the business domain. It distills and organizes domain knowledge, and provides a common language for developers and domain experts.

Start by mapping all of the business functions and their connections. This will likely be a collaborative effort that involves domain experts, software architects, and other stakeholders. You don't need to use any particular formalism. Sketch a diagram or draw on whiteboard.

As you fill in the diagram, you may start to identify discrete subdomains. Which functions are closely related? Which functions are core to the business, and which provide ancillary services? What is the dependency graph? During this initial phase, you aren't concerned with technologies or implementation details. That said, you should note the place where the application will need to integrate with external systems, such as CRM, payment processing, or billing systems.

Example: Drone delivery application

After some initial domain analysis, the Fabrikam team came up with a rough sketch that depicts the Drone Delivery domain.



- **Shipping** is placed in the center of the diagram, because it's core to the business. Everything else in the diagram exists to enable this functionality.
- **Drone management** is also core to the business. Functionality that is closely related to drone management includes **drone repair** and using **predictive analysis** to predict when drones need servicing and maintenance.
- **ETA analysis** provides time estimates for pickup and delivery.
- **Third-party transportation** will enable the application to schedule alternative transportation methods if a package cannot be shipped entirely by drone.
- **Drone sharing** is a possible extension of the core business. The company may have excess drone capacity during certain hours, and could rent out drones that would otherwise be idle. This feature will not be in the initial release.
- **Video surveillance** is another area that the company might expand into later.
- **User accounts**, **Invoicing**, and **Call center** are subdomains that support the core business.

Notice that at this point in the process, we haven't made any decisions about implementation or technologies. Some of the subsystems may involve external software systems or third-party services. Even so, the application needs to interact with these systems and services, so it's important to include them in the domain model.

NOTE

When an application depends on an external system, there is a risk that the external system's data schema or API will leak into your application, ultimately compromising the architectural design. This is particularly true with legacy systems that may not follow modern best practices, and may use convoluted data schemas or obsolete APIs. In that case, it's important to have a well-defined boundary between these external systems and the application. Consider using the [Strangler Fig pattern](#) or the [Anti-Corruption Layer pattern](#) for this purpose.

Define bounded contexts

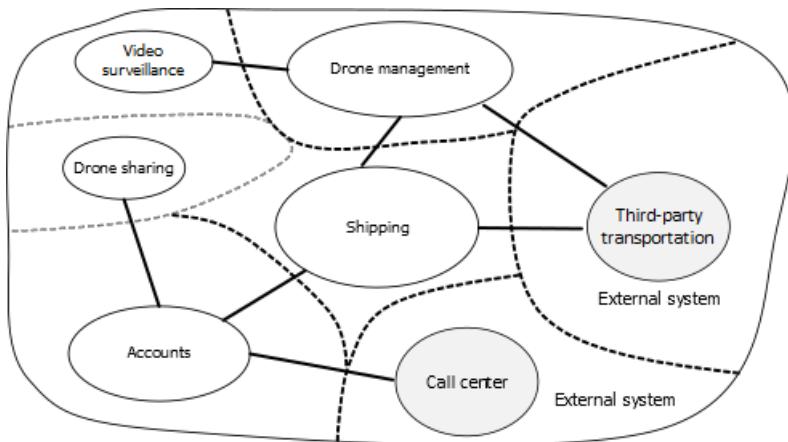
The domain model will include representations of real things in the world — users, drones, packages, and so forth. But that doesn't mean that every part of the system needs to use the same representations for the same things.

For example, subsystems that handle drone repair and predictive analysis will need to represent many physical characteristics of drones, such as their maintenance history, mileage, age, model number, performance characteristics, and so on. But when it's time to schedule a delivery, we don't care about those things. The scheduling subsystem only needs to know whether a drone is available, and the ETA for pickup and delivery.

If we tried to create a single model for both of these subsystems, it would be unnecessarily complex. It would also become harder for the model to evolve over time, because any changes will need to satisfy multiple teams working on separate subsystems. Therefore, it's often better to design separate models that represent the same real-world entity (in this case, a drone) in two different contexts. Each model contains only the features and

attributes that are relevant within its particular context.

This is where the DDD concept of *bounded contexts* comes into play. A bounded context is simply the boundary within a domain where a particular domain model applies. Looking at the previous diagram, we can group functionality according to whether various functions will share a single domain model.



Bounded contexts are not necessarily isolated from one another. In this diagram, the solid lines connecting the bounded contexts represent places where two bounded contexts interact. For example, Shipping depends on User Accounts to get information about customers, and on Drone Management to schedule drones from the fleet.

In the book *Domain Driven Design*, Eric Evans describes several patterns for maintaining the integrity of a domain model when it interacts with another bounded context. One of the main principles of microservices is that services communicate through well-defined APIs. This approach corresponds to two patterns that Evans calls Open Host Service and Published Language. The idea of Open Host Service is that a subsystem defines a formal protocol (API) for other subsystems to communicate with it. Published Language extends this idea by publishing the API in a form that other teams can use to write clients. In the article [Designing APIs for microservices](#), we discuss using [OpenAPI Specification](#) (formerly known as Swagger) to define language-agnostic interface descriptions for REST APIs, expressed in JSON or YAML format.

For the rest of this journey, we will focus on the Shipping bounded context.

Next steps

After completing a domain analysis, the next step is to apply tactical DDD, to define your domain models with more precision.

[Tactical DDD](#)

Choose an Azure compute option for microservices

3/10/2022 • 4 minutes to read • [Edit Online](#)

The term *compute* refers to the hosting model for the computing resources that your application runs on. For a microservices architecture, two approaches are especially popular:

- A service orchestrator that manages services running on dedicated nodes (VMs).
- A serverless architecture using functions as a service (FaaS).

While these aren't the only options, they are both proven approaches to building microservices. An application might include both approaches.

Service orchestrators

An orchestrator handles tasks related to deploying and managing a set of services. These tasks include placing services on nodes, monitoring the health of services, restarting unhealthy services, load balancing network traffic across service instances, service discovery, scaling the number of instances of a service, and applying configuration updates. Popular orchestrators include Kubernetes, Service Fabric, DC/OS, and Docker Swarm.

On the Azure platform, consider the following options:

- [Azure Kubernetes Service](#) (AKS) is a managed Kubernetes service. AKS provisions Kubernetes and exposes the Kubernetes API endpoints, but hosts and manages the Kubernetes control plane, performing automated upgrades, automated patching, autoscaling, and other management tasks. You can think of AKS as being "Kubernetes APIs as a service."
- [Service Fabric](#) is a distributed systems platform for packaging, deploying, and managing microservices. Microservices can be deployed to Service Fabric as containers, as binary executables, or as [Reliable Services](#). Using the Reliable Services programming model, services can directly use Service Fabric programming APIs to query the system, report health, receive notifications about configuration and code changes, and discover other services. A key differentiation with Service Fabric is its strong focus on building stateful services using [Reliable Collections](#).
- Other options such as Docker Enterprise Edition and Mesosphere DC/OS can run in an IaaS environment on Azure. You can find deployment templates on [Azure Marketplace](#).

Containers

Sometimes people talk about containers and microservices as if they were the same thing. While that's not true — you don't need containers to build microservices — containers do have some benefits that are particularly relevant to microservices, such as:

- **Portability.** A container image is a standalone package that runs without needing to install libraries or other dependencies. That makes them easy to deploy. Containers can be started and stopped quickly, so you can spin up new instances to handle more load or to recover from node failures.
- **Density.** Containers are lightweight compared with running a virtual machine, because they share OS resources. That makes it possible to pack multiple containers onto a single node, which is especially useful when the application consists of many small services.
- **Resource isolation.** You can limit the amount of memory and CPU that is available to a container, which can help to ensure that a runaway process doesn't exhaust the host resources. See the [Bulkhead pattern](#) for more information.

Serverless (Functions as a Service)

With a [serverless](#) architecture, you don't manage the VMs or the virtual network infrastructure. Instead, you deploy code and the hosting service handles putting that code onto a VM and executing it. This approach tends to favor small granular functions that are coordinated using event-based triggers. For example, a message being placed onto a queue might trigger a function that reads from the queue and processes the message.

[Azure Functions](#) is a serverless compute service that supports various function triggers, including HTTP requests, Service Bus queues, and Event Hubs events. For a complete list, see [Azure Functions triggers and bindings concepts](#). Also consider [Azure Event Grid](#), which is a managed event routing service in Azure.

Orchestrator or serverless?

Here are some factors to consider when choosing between an orchestrator approach and a serverless approach.

Manageability A serverless application is easy to manage, because the platform manages all the of compute resources for you. While an orchestrator abstracts some aspects of managing and configuring a cluster, it does not completely hide the underlying VMs. With an orchestrator, you will need to think about issues such as load balancing, CPU and memory usage, and networking.

Flexibility and control. An orchestrator gives you a great deal of control over configuring and managing your services and the cluster. The tradeoff is additional complexity. With a serverless architecture, you give up some degree of control because these details are abstracted.

Portability. All of the orchestrators listed here (Kubernetes, DC/OS, Docker Swarm, and Service Fabric) can run on-premises or in multiple public clouds.

Application integration. It can be challenging to build a complex application using a serverless architecture, due to the need to coordinate, deploy, and manage many small independent functions. One option in Azure is to use [Azure Logic Apps](#) to coordinate a set of Azure Functions. For an example of this approach, see [Create a function that integrates with Azure Logic Apps](#).

Cost. With an orchestrator, you pay for the VMs that are running in the cluster. With a serverless application, you pay only for the actual compute resources consumed. In both cases, you need to factor in the cost of any additional services, such as storage, databases, and messaging services.

Scalability. Azure Functions scales automatically to meet demand, based on the number of incoming events. With an orchestrator, you can scale out by increasing the number of service instances running in the cluster. You can also scale by adding additional VMs to the cluster.

Our reference implementation primarily uses Kubernetes, but we did use Azure Functions for one service, namely the Delivery History service. Azure Functions was a good fit for this particular service, because it's an event-driven workload. By using an Event Hubs trigger to invoke the function, the service needed a minimal amount of code. Also, the Delivery History service is not part of the main workflow, so running it outside of the Kubernetes cluster doesn't affect the end-to-end latency of user-initiated operations.

Next steps

[Interservice communication](#)

Serverless functions architecture design

3/10/2022 • 2 minutes to read • [Edit Online](#)

Serverless architecture evolves cloud platforms toward pure cloud-native code by abstracting code from the infrastructure that it needs to run. [Azure Functions](#) is a serverless compute option that supports *functions*, small pieces of code that do single things.

Benefits of using serverless architectures with Functions applications include:

- The Azure infrastructure automatically provides all the updated servers that applications need to keep running at scale.
- Compute resources allocate dynamically, and instantly autoscale to meet elastic demands. Serverless doesn't mean "no server," but "less server," because servers run only as needed.
- Micro-billing saves costs by charging only for the compute resources and duration the code uses to execute.
- Function *bindings* streamline integration by providing declarative access to a wide variety of Azure and third-party services.

Functions are *event-driven*. An external event like an HTTP web request, message, schedule, or change in data triggers the function code. A Functions application doesn't code the trigger, only the response to the trigger. With a lower barrier to entry, developers can focus on business logic, rather than writing code to handle infrastructure concerns like messaging.

Azure Functions is a managed service in Azure and Azure Stack. The open source Functions runtime works in many environments, including Kubernetes, Azure IoT Edge, on-premises, and other clouds.

Serverless and Functions require new ways of thinking and new approaches to building applications. They aren't the right solutions for every problem. For example serverless Functions scenarios, see [Reference architectures](#).

Implementation steps

Successful implementation of serverless technologies with Azure Functions requires the following actions:

- [Decide and plan](#)

Architects and technical decision makers (TDMs) perform [application assessment](#), conduct or attend [technical workshops and trainings](#), run [proof of concept \(PoC\) or pilot](#) projects, and conduct architectural designs sessions as necessary.

- [Develop and deploy apps](#)

Developers implement serverless Functions app development patterns and practices, configure DevOps pipelines, and employ site reliability engineering (SRE) best practices.

- [Manage operations](#)

IT professionals identify hosting configurations, future-proof scalability by automating infrastructure provisioning, and maintain availability by planning for business continuity and disaster recovery.

- [Secure apps](#)

Security professionals handle Azure Functions security essentials, secure the hosting setup, and provide application security guidance.

Related resources

- To learn more about serverless technology, see the [Azure serverless documentation](#).
- To learn more about Azure Functions, see the [Azure Functions documentation](#).
- For help with choosing a compute technology, see [Choose an Azure compute service for your application](#).

Serverless Functions reference architectures

3/10/2022 • 4 minutes to read • [Edit Online](#)

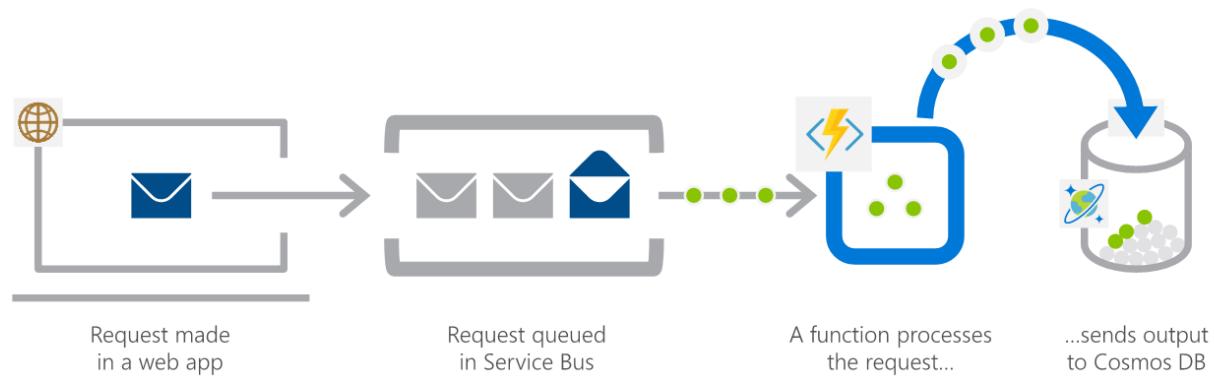
A reference architecture is a template of required components and the technical requirements to implement them. A reference architecture isn't custom-built for a customer solution, but is a high-level scenario based on extensive experience. Before designing a serverless solution, use a reference architecture to visualize an ideal technical architecture, then blend and integrate it into your environment.

Common serverless architecture patterns

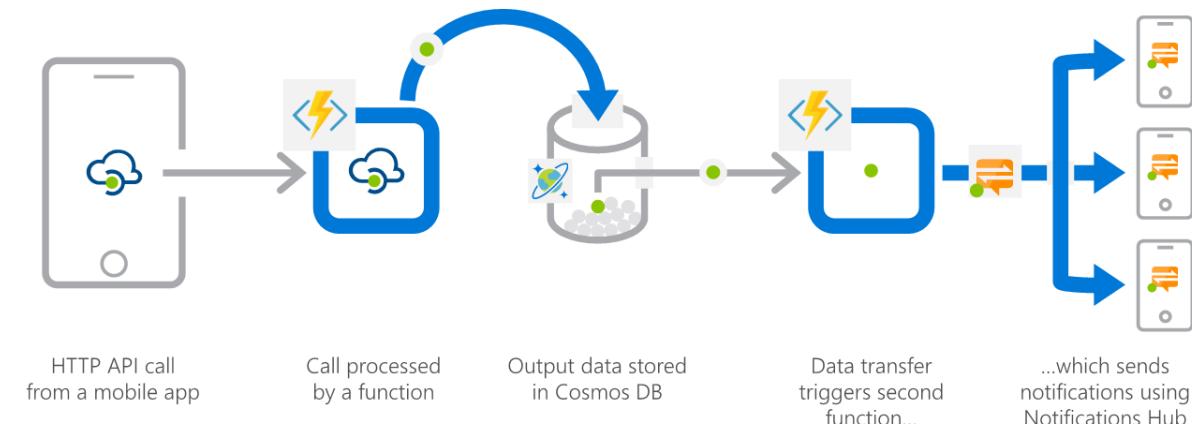
Common serverless architecture patterns include:

- Serverless APIs, mobile and web backends.
- Event and stream processing, Internet of Things (IoT) data processing, big data and machine learning pipelines.
- Integration and enterprise service bus to connect line-of-business systems, publish and subscribe (Pub/Sub) to business events.
- Automation and digital transformation and process automation.
- Middleware, software-as-a-Service (SaaS) like Dynamics, and big data projects.

Web application backends Retail scenario: Pick up online orders from a queue, process them, and store the resulting data in a database



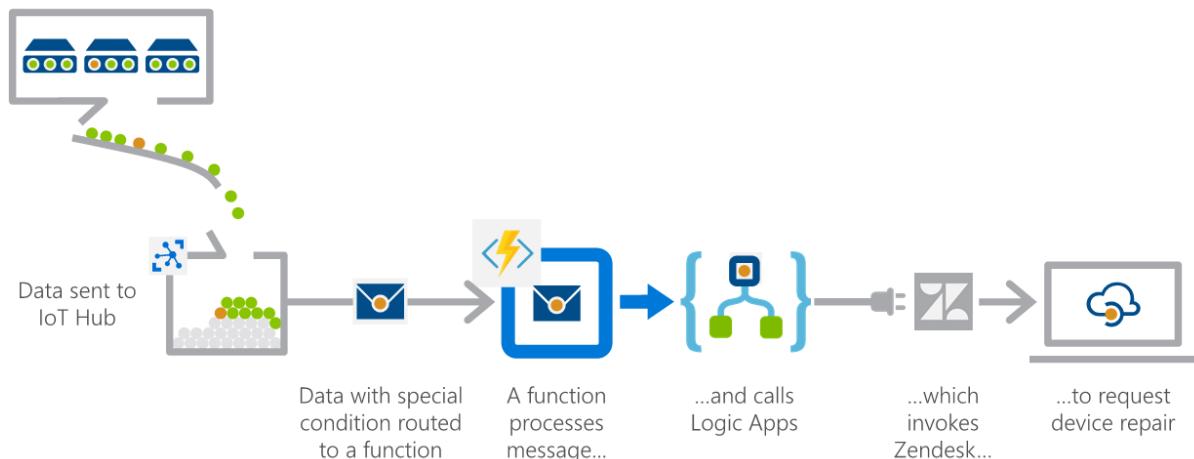
Mobile application backends Financial services scenario: Colleagues use mobile banking to reimburse each other for lunch. Whoever paid for lunch requests payment through a mobile app, triggering a notification on colleagues' phones.



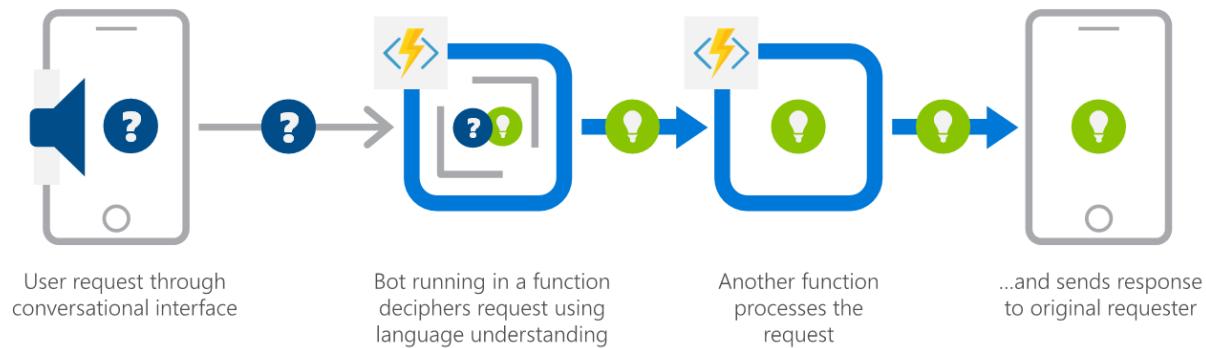
IoT-connected backends Manufacturing scenario: A manufacturing company uses IoT to monitor its machines. Functions detects anomalous data and triggers a message to the service department when repair is

required.

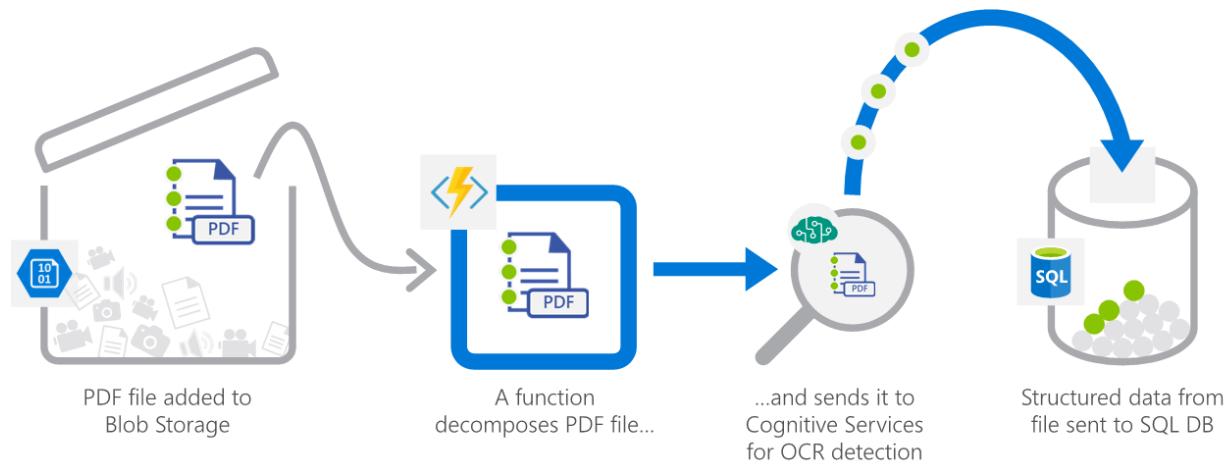
Connected IoT devices producing data



Conversational bot processing Hospitality scenario: Customers ask for available vacation accommodations on their smartphones. A serverless bot deciphers requests and returns vacation options.

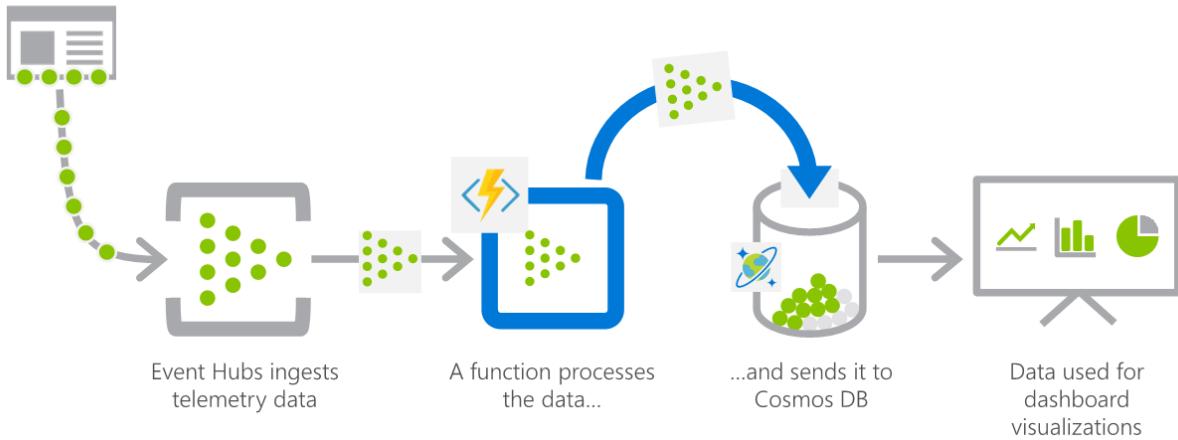


Real-time file processing Healthcare scenario: The solution securely uploads patient records as PDF files. The solution then decomposes the data, processes it using OCR detection, and adds it to a database for easy queries.

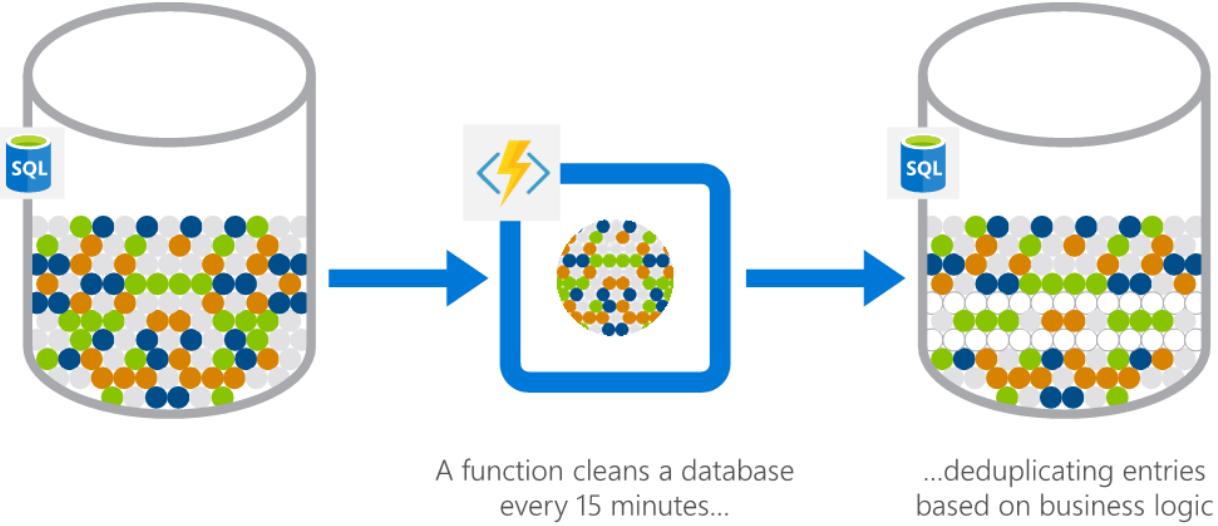


Real-time stream processing Independent software vendor (ISV) scenario: A massive cloud app collects huge amounts of telemetry data. The app processes that data in near real-time and stores it in a database for use in an analytics dashboard.

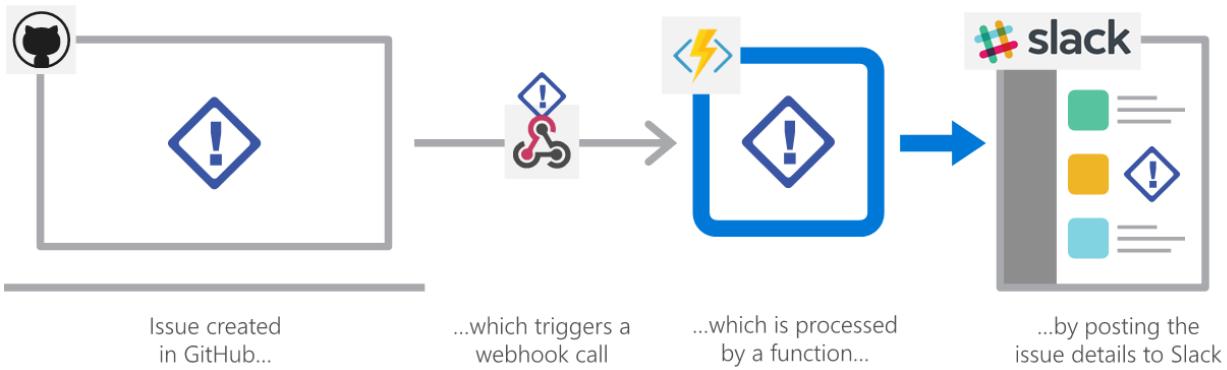
App or device producing data



Scheduled task automation Financial services scenario: The app analyzes a customer database for duplicate entries every 15 minutes, to avoid sending out multiple communications to the same customers.



Extending SaaS applications Professional services scenario: A SaaS solution provides extensibility through webhooks, which Functions can implement to automate certain workflows.



Featured serverless reference architectures

The following featured serverless reference architectures walk through specific scenarios. See the linked articles for architectural diagrams and details.

Serverless microservices

The [serverless microservices reference architecture](#) walks you through designing, developing, and delivering the Rideshare application by Relecloud, a fictitious company. You get hands-on instructions for configuring and deploying all the architectural components, with helpful information about each component.

Serverless web application and event processing with Azure Functions

This two-part solution describes a hypothetical drone delivery system. Drones send in-flight status to the cloud, which stores these messages for later use. A web application allows users to retrieve the messages to get the latest device status.

- You can download the code for this solution from [GitHub](#).
- The article [Code walkthrough: Serverless application with Azure Functions](#) walks you through the code and the design processes.

Event-based cloud automation

Automating workflows and repetitive tasks on the cloud can dramatically improve a DevOps team's productivity. A serverless model is best suited for event-driven automation scenarios. This [event-based automation reference architecture](#) illustrates two cloud automation scenarios: cost center tagging and throttling response.

Multicloud with Serverless Framework

The [Serverless Framework architecture](#) describes how the Microsoft Commercial Software Engineering (CSE) team partnered with a global retailer to deploy a highly-available serverless solution across both Azure and Amazon Web Services (AWS) cloud platforms, using the Serverless Framework.

More serverless Functions reference architectures

The following sections list other serverless and Azure Functions-related reference architectures and scenarios.

General

- [Serverless application architectures using Event Grid](#)
- [Serverless apps using Cosmos DB](#)
- [Serverless event processing using Azure Functions](#)
- [Serverless web application on Azure](#)
- [Serverless Asynchronous Multiplayer Reference Architecture](#)
- [Instant Broadcasting on Serverless Architecture](#)
- [Building a telehealth system on Azure](#)
- [Custom Data Sovereignty & Data Gravity Requirements](#)
- [Sharing location in real time using low-cost serverless Azure services](#)

Web and mobile backend

- [An e-commerce front end](#)
- [Architect scalable e-commerce web app](#)
- [Improve scalability in an Azure web application](#)
- [Uploading and CDN-preloading static content with Azure Functions](#)
- [Cross Cloud Scaling Architecture](#)
- [Social App for Mobile and Web with Authentication](#)

AI + Machine Learning

- [Image classification for insurance claims](#)
- [Personalized Offers](#)
- [Personalized marketing solutions](#)
- [Speech transcription with Azure Cognitive Services](#)
- [Training a Model with AzureML and Azure Functions](#)
- [Customer Reviews App with Cognitive Services](#)
- [Enterprise-grade conversational bot](#)
- [AI at the Edge](#)

- Mass ingestion and analysis of news feeds on Azure
- HIPPA and HITRUST compliant health data AI
- Intelligent Experiences On Containers (AKS, Functions, Keda)

Data and analytics

- Application integration using Event Grid
- Mass ingestion and analysis of news feeds
- Tier Applications & Data for Analytics
- Operational analysis and driving process efficiency

IoT

- Azure IoT reference (SQL DB)
- Azure IoT reference (Cosmos DB)
- IoT using Cosmos DB
- Facilities management powered by mixed reality and IoT
- Complementary Code Pattern for Azure IoT Edge Modules & Cloud Applications

Gaming

- Custom Game Server Scaling
- Non-real Time Dashboard
- In-editor Debugging Telemetry
- Multiplayer Serverless Matchmaker
- Advanced leaderboard for large scale
- Relational Leaderboard
- Content Moderation
- Text Translation
- Text to Speech
- Gaming using Cosmos DB

Automation

- Smart scaling for Azure Scale Set with Azure Functions

Plan for deploying serverless Functions

3/10/2022 • 2 minutes to read • [Edit Online](#)

To plan for moving an application to a serverless Azure Functions architecture, a technical decision maker (TDM) or architect:

- Verifies the application's characteristics and business requirements.
- Determines the application's suitability for serverless Azure Functions.
- Transforms business requirements into functional and other requirements.

Planning activities also include assessing technical team readiness, providing or attending workshops and training, and conducting architectural design reviews, proofs of concept, pilots, and technical implementations.

TDMs and architects may perform one or more of the following activities:

- **Execute an application assessment.** Evaluate the main aspects of the application to determine how complex and risky it is to rearchitect through application modernization, or rebuild a new cloud-native application. See [Application assessment](#).
- **Attend or promote technical workshops and training.** Host a Serverless workshop or CloudHack, or enjoy many other training and learning opportunities for serverless technologies, Azure Functions, app modernization, and cloud-native apps. See [Technical workshops and training](#).
- **Identify and execute a Proof of Concept (PoC) or pilot, or technical implementation.** Deliver a PoC, pilot, or technical implementation to provide evidence that serverless Azure Functions can solve a team's business problems. Showing teams how to modernize or build new cloud-native applications to their specifications can accelerate deployment to production. See [PoC or pilot](#).
- **Conduct architectural design sessions.** An architectural design session (ADS) is an in-depth discussion on how a new solution will blend into the environment. ADSs validate business requirements and transform them to functional and other requirements.

Next steps

For example scenarios that use serverless architectures with Azure Functions, see [Serverless reference architectures](#).

To move forward with serverless Azure Functions implementation, see the following resources:

- [Application development and deployment](#)
- [Azure Functions app operations](#)
- [Azure Functions app security](#)

Application assessment

3/10/2022 • 6 minutes to read • [Edit Online](#)

Cloud rationalization is the process of evaluating applications to determine the best way to migrate or modernize them for the cloud.

Rationalization methods include:

- **Rehost.** Also known as a *lift and shift* migration, rehost moves a current application to the cloud with minimal change.
- **Refactor.** Slightly refactoring an application to fit *platform-as-a-service* (PaaS)-based options can reduce operational costs.
- **Rearchitect.** Rearchitect aging applications that aren't compatible with cloud components, or cloud-compatible applications that would realize cost and operational efficiencies by rearchitecting into a cloud-native solution.
- **Rebuild.** If the changes or costs to carry an application forward are too great, consider creating a new cloud-native code base. Rebuild is especially appropriate for applications that previously met business needs, but are now unsupported or misaligned with current business processes.

Before you decide on an appropriate strategy, analyze the current application to determine the risk and complexity of each method. Consider application lifecycle, technology, infrastructure, performance, and operations and monitoring. For multitier architectures, evaluate the presentation tier, service tier, integrations tier, and data tier.

The following checklists evaluate an application to determine the complexity and risk of rearchitecting or rebuilding.

Complexity and risk

Each of the following factors adds to complexity, risk, or both.

Architecture

Define the high-level architecture, such as web application, web services, data storage, or caching.

FACTOR	COMPLEXITY	RISK
Application components don't translate directly to Azure.	✓	✓
The application needs code changes to run in Azure.	✓	✓
The application needs major, complex code changes to run in Azure.	✓	✓

Business drivers

Older applications might require extensive changes to get to the cloud.

FACTOR	COMPLEXITY	RISK
This application has been around for more than three years.	✓	
This application is business critical.		✓
There are technology blockers for migration.	✓	
There are business blockers for migration.	✓	
This application has compliance requirements.		✓
The application is subject to country-specific data requirements.		✓
The application is publicly accessible.	✓	✓

Technology

FACTOR	COMPLEXITY	RISK
This is not a web-based application, and isn't hosted on a web server.	✓	
The app isn't hosted in Windows IIS	✓	
The app isn't hosted on Linux	✓	
The application is hosted in a web farm, and requires multiple servers to host the web components.	✓	✓
The application requires third-party software to be installed on the servers.	✓	✓
The application is hosted in a single datacenter, and operations are performed in a single location.	✓	
The application accesses the server's registry.	✓	
The application sends emails, and needs access to an SMTP server.	✓	
This isn't a .NET application.	✓	
The application uses SQL Server as its data store.	✓	

FACTOR	COMPLEXITY	RISK
The application stores data on local disks, and needs access to the disks to operate properly.	✓	
The application uses Windows Services to process asynchronous operations, or needs external services to process data or operations.	✓	

Deployment

When assessing deployment requirements, consider:

- Number of daily users
- Average number of concurrent users
- Expected traffic
- Bandwidth in Gbps
- Requests per second
- Amount of memory needed

You can reduce deployment risk by storing code under source control in a version control system such as Git, Azure DevOps Server, or SVN.

FACTOR	COMPLEXITY	RISK
Leveraging existing code and data is a #1 priority.	✓	✓
The application code isn't under source control.		✓
There's no automated build process like Azure DevOps Server or Jenkins.		✓
There's no automated release process to deploy the application.	✓	✓
The application has a Service Level Agreement (SLA) that dictates the amount of expected downtime.		✓
The application experiences peak or variable usage times or loads.		✓
The web application saves its session state in process, rather than an external data store.	✓	

Operations

FACTOR	COMPLEXITY	RISK
The application doesn't have a well-established instrumentation strategy or standard instrumentation framework.		✓
The application doesn't use monitoring tools, and the operations team doesn't monitor the app's performance.		✓
The application has measured SLA in place, and the operations team monitors the application's performance.		✓
The application writes to a log store, event log, log file, log database, or Application Insights.	✓	
The application doesn't write to a log store, event log, log file, log database, or Application Insights.		✓
The application isn't part of the organization's disaster recovery plan.		✓

Security

FACTOR	COMPLEXITY	RISK
The application uses Active Directory to authenticate users.	✓	✓
The organization hasn't yet configured Azure Active Directory (Azure AD), or hasn't configured Azure AD Connect to synchronize on-premises AD with Azure AD.	✓	
The application requires access to on-premises resources, which will require VPN connectivity from Azure.	✓	
The organization hasn't yet configured a VPN connection between Azure and their on-premises environment.	✓	✓
The application requires a SSL certificate to run.	✓	✓

Results

Count your application's Complexity and Risk checkmarks.

- The expected level of complexity to migrate or modernize the application to Azure is: **Total Complexity/25**.
- The expected risk involved is: **Total Risk/19**.

For both complexity and risk, a score of <0.3 = low, <0.7 = medium, >0.7 = high.

Refactor, rearchitect, or rebuild

To rationalize whether to rehost, refactor, rearchitect, or rebuild your application, consider the following points. Many of these factors also contribute to complexity and risk.

Determine whether the application components can translate directly to Azure. If so, you don't need code changes to move the application to Azure, and could use rehost or refactor strategies. If not, you need to rewrite code, so you need to rearchitect or rebuild.

If the app does need code changes, determine the complexity and extent of the needed changes. Minor changes might allow for rearchitecting, while major changes may require rebuilding.

Rehost or refactor

- If leveraging existing code and data is a top priority, consider a refactor strategy rather than rearchitecting or rebuilding.
- If you have pressing timelines like datacenter shutdown or contract expiration, end-of-life licensing, or mergers or acquisitions, the fastest way to get the application to Azure might be to rehost, followed by refactoring to take advantage of cloud capabilities.

Rearchitect or rebuild

- If there are applications serving similar needs in your portfolio, this might be an opportunity to rearchitect or rebuild the entire solution.
- If you want to [implement multi-tier or microservices architecture](#) for a monolithic app, you must rearchitect or rebuild the app. If you don't mind retaining the monolithic structure, you might be able to rehost or refactor.
- Rearchitect or rebuild the app to take advantage of cloud capabilities if you plan to update the app more often than yearly, if the app has peak or variable usage times, or if you expect the app to handle high traffic.

To decide between rearchitecting or rebuilding, assess the following factors. The largest scoring result indicates your best strategy.

FACTOR	REARCHITECT	REBUILD
There are other applications serving similar needs in your portfolio.	✓	✓
The application needs minor code changes to run in Azure.	✓	
The application needs major, complex code changes to run in Azure.		✓
It's important to leverage existing code.	✓	
You want to move a monolithic application to multi-tier architecture.	✓	

FACTOR	RARCHITECT	REBUILD
You want to move a monolithic application to a microservices architecture.	✓	✓
You expect this app to add breakthrough capabilities like AI, IoT, or bots.		✓
Among functionality, cost, infrastructure, and processes, functionality is the least efficient aspect of this application.		✓
The application requires third-party software installed on the servers.	✓	
The application accesses the server's registry.	✓	
The application sends emails and needs access to an SMTP server.	✓	
The application uses SQL Server as its data store.	✓	
The application stores data on local disks, and needs access to the disks to run properly.	✓	
The application uses Windows services to process asynchronous operations, or needs external services to process data or operations.	✓	
A web application saves its session state in process, rather than to an external data store.	✓	
The app has peak and variable usage times and loads.	✓	✓
You expect the application to handle high traffic.	✓	✓

Technical workshops and training

3/10/2022 • 3 minutes to read • [Edit Online](#)

The workshops, classes, and learning materials in this article provide technical training for serverless architectures with Azure Functions. These resources help you and your team or customers understand and implement application modernization and cloud-native apps.

Technical workshops

The [Microsoft Cloud Workshop \(MCW\)](#) program provides workshops you can host to foster cloud learning and adoption. Each workshop includes presentation decks, trainer and student guides, and hands-on lab guides. Contribute your own content and feedback to add to a robust database of training guides for deploying advanced Azure workloads on the Microsoft Cloud Platform.

Workshops related to application development workloads include:

- [Serverless APIs in Azure](#). Set of entry-level exercises, which cover the basics of building and managing serverless APIs in Microsoft Azure - with Azure Functions, Azure API Management, and Azure Application Insights.
- [Serverless architecture](#). Implement a series of Azure Functions that independently scale and break down business logic to discrete components, allowing customers to pay only for the services they use.
- [App modernization](#). Design a modernization plan to move services from on-premises to the cloud by leveraging cloud, web, and mobile services, secured by Azure Active Directory.
- [Modern cloud apps](#). Deploy, configure, and implement an end-to-end secure and Payment Card Industry (PCI) compliant solution for e-commerce, based on Azure App Services, Azure Active Directory, and Azure DevOps.
- [Cloud-native applications](#). Using DevOps best practices, build a proof of concept (PoC) to transform a platform-as-a-service (PaaS) application to a container-based application with multi-tenant web app hosting.
- [Continuous delivery in Azure DevOps](#). Set up and configure continuous delivery (CD) in Azure to reduce manual errors, using Azure Resource Manager templates, Azure DevOps, and Git repositories for source control.

Instructor-led training

[Course AZ-204: Developing solutions for Microsoft Azure](#) teaches developers how to create end-to-end solutions in Microsoft Azure. Students learn how to implement Azure compute solutions, create Azure Functions, implement and manage web apps, develop solutions utilizing Azure Storage, implement authentication and authorization, and secure their solutions by using Azure Key Vault and managed identities. Students also learn to connect to and consume Azure and third-party services, and include event- and message-based models in their solutions. The course also covers monitoring, troubleshooting, and optimizing Azure solutions.

Serverless OpenHack

The Serverless [OpenHack](#) simulates a real-world scenario where a company wants to utilize serverless services to build and release an API to integrate into their distributor's application. This OpenHack lets attendees quickly build and deploy Azure serverless solutions with cutting-edge compute services like Azure Functions, Logic Apps, Event Grid, Service Bus, Event Hubs, and Cosmos DB. The OpenHack also covers related technologies like API Management, Azure DevOps or GitHub, Application Insights, Dynamics 365/Microsoft 365, and Cognitive APIs.

OpenHack attendees build serverless functions, web APIs, and a CI/CD pipeline to support them. They implement further serverless technologies to integrate line of business (LOB) app workflows, process user and data telemetry, and create key progress indicator (KPI)-aligned reports. By the end of the OpenHack, attendees have built out a full serverless technical solution that can create workflows between systems and handle events, files, and data ingestion.

Microsoft customer projects inspired these OpenHack challenges:

- Configure the developer environment.
- Create your first serverless function and workflow.
- Build APIs to support business needs.
- Deploy a management layer for APIs and monitoring usage.
- Build a LOB workflow process.
- Process large amounts of unstructured file data.
- Process large amounts of incoming event data.
- Implement a publisher/subscriber messaging pattern and virtual network integration.
- Conduct sentiment analysis.
- Perform data aggregation, analysis, and reporting.

To attend an OpenHack, register at <https://openhack.microsoft.com>. For enterprises with many engineers, Microsoft can request and organize a dedicated Serverless OpenHack.

Microsoft Learn

[Microsoft Learn](#) is a free, online training platform that provides interactive learning for Microsoft products. The goal is to improve proficiency with fun, guided, hands-on content that's specific to your role and goals. Learning paths are collections of modules that are organized around specific roles like developer, architect, or system admin, or technologies like Azure Web Apps, Azure Functions, or Azure SQL DB. Learning paths provide understanding of different aspects of the technology or role.

Learning paths about serverless apps and Azure Functions include:

- [Create serverless applications](#). Learn how to leverage functions to execute server-side logic and build serverless architectures.
- [Architect message brokering and serverless applications in Azure](#). Learn how to create reliable messaging for your applications, and how to take advantage of serverless application services in Azure.
- [Search all Functions-related learning paths](#).

Hands-on labs and how-to guides

- [Build a serverless web app](#), from the Build 2018 conference
- [Build a Serverless IoT Solution with Python Azure Functions and SignalR](#)

Next steps

- [Execute an application assessment](#)
- [Identify and execute a PoC or Pilot project](#)

Proof of concept or pilot

3/10/2022 • 4 minutes to read • [Edit Online](#)

When driving a technical and security decision for your company or customer, a *Proof of Concept (PoC)* or *pilot* is an opportunity to deliver evidence that the proposed solution solves the business problems. The PoC or pilot increases the likelihood of a successful adoption.

A PoC:

- Demonstrates that a business model or idea is feasible and will work to solve the business problem
- Usually involves one to three features or capabilities
- Can be in one or multiple technologies
- Is usually geared toward a particular scenario, and proves what the customer needs to know to make the technical or security decision
- Is used only as a demonstration and won't go into production
- Is IT-driven and enablement-driven

A pilot:

- Is a test run or trial of a proposed action or product
- Lasts longer than a PoC, often weeks or months
- Has a higher return on investment (ROI) than a PoC
- Builds in a pre-production or trial environment, with the intent that it will then go into production
- Is adoption-driven and consumption-driven

PoC and pilot best practices

Be aware of compliance issues when working in a customer's environment, and make sure your actions are always legal and compliant.

- Touching or altering the customer's environment usually requires a contract, and may involve a partner or Microsoft services. Without a contract, your company may be liable for issues or damages.
- Governance may require Legal department approval. Your company may not be able to give intellectual property (IP) away for free. You may need a legal contract or contracts to specify whether your company or the customer pays for the IP.
- Get disclosure guidance when dealing with non-disclosure agreements (NDAs), product roadmaps, NDA features, or anything not released to the general public.
- In a pilot, don't use a trial Microsoft Developers Network (MSDN) environment, or any environment that you own.
- Use properly-licensed software, and ask the opportunity owner to make sure to handle software licensing correctly.

The customer, partner, or your company may pay for the PoC or pilot. Depending on the size of the contract, the ROI, and the cost of sale, one group may cover it all, or a combination of all three parties may cover the cost. Ensure that your company or customer has some investment in the PoC or pilot. If they don't, this can be a red flag signaling that your company or customer doesn't yet see value in the solution.

PoC and pilot process

The Technical Decision Maker (TDM) is responsible for driving an adoption decision. The TDM is responsible for

ensuring that the right partners and resources are involved in a PoC or pilot. As a TDM, make sure you're aware of the partners in your product and service area or region. Be aware of their key service offerings around your product service area.

Planning

Consider the following health questions:

- Do you have a good technical plan, including key decision makers and Microsoft potential?
- Can you deliver the needed assurance without a PoC?
- Should you switch to a pilot?
- What are the detailed scope and decision criteria your team or customer agreed to?
- If you meet the criteria, will your company or customer buy or deploy the solution?

Do the following tasks:

- Analyze risk.
- Evaluate the setting.
- Do the preparation.
- Consider workloads and human resources.
- Present PoC or pilot health status.
- Fulfill technical prerequisites.
- Define the go/no go decision.
- Create a final project plan specification.

Execution

For the execution phase:

- Determine who kicks off the presentation.
- Schedule the meeting in the morning, if possible.
- Prepare the demos and slides.
- Conduct a dry run to refine the presentation.
- Get feedback.
- Involve your company or customer team.
- Complete the win/lose statement.

Debriefing

During the debriefing phase, consider:

- Whether criteria were met or not met
- Stakeholder investment
- Initiating deployment
- Finding a partner and training
- Lessons learned
- Corrections or extensions of PoC or pilot guidance
- Archiving of valuable deliverables

Change management

Change management uses tested methods and techniques to avoid errors and minimize impact when administering change.

Ideally, a pilot includes a cross-section of users, to address any potential issues or problems that arise. Users may be comfortable and familiar with their old technology, and have difficulty moving into new technical

solutions. Change management keeps this in mind, and helps the user understand the reasons behind the change and the impact the change will make.

This understanding is part of a pilot, and addresses everyone who has a stake in the project. A pilot is better than a PoC, because the customer is more involved, so they're more likely to implement the change.

The pilot includes a detailed follow up through surveys or focus groups. The feedback can prove and improve the change.

Next steps

- [Execute an application assessment](#)
- [Promote a technical workshop or training](#)
- [Code a technical implementation with the team or customer](#)

Related resources

[Prosci® change management training](#)

Application development and deployment

3/10/2022 • 4 minutes to read • [Edit Online](#)

To develop and deploy serverless applications with Azure Functions, examine patterns and practices, configure DevOps pipelines, and implement site reliability engineering (SRE) best practices.

For detailed information about serverless architectures and Azure Functions, see:

- [Serverless apps: Architecture, patterns, and Azure implementation](#)
- [Azure Serverless Computing Cookbook](#)
- [Example serverless reference architectures](#)

Planning

To plan app development and deployment:

1. Prepare development environment and set up workflow.
2. Structure projects to support Azure Functions app development.
3. Identify app triggers, bindings, and configuration requirements.

Understand event-driven architecture

A different event triggers every function in a serverless Functions project. For more information about event-driven architectures, see:

- [Event-driven architecture style](#).
- [Event-driven design patterns to enhance existing applications using Azure Functions](#)

Prepare development environment

Set up your development workflow and environment with the tools to create Functions. For details about development tools and Functions code project structure, see:

- [Code and test Azure Functions locally](#)
- [Develop Azure Functions by using Visual Studio Code](#)
- [Develop Azure Functions using Visual Studio](#)
- [Work with Azure Functions Core Tools](#)
- [Folder structure](#)

Development

Decide on the development language to use. Azure Functions supports C#, F#, PowerShell, JavaScript, TypeScript, Java, and Python. All of a project's Functions must be in the same language. For more information, see [Supported languages in Azure Functions](#).

Define triggers and bindings

A trigger invokes a Function, and every Function must have exactly one trigger. Binding to a Function declaratively connects another resource to the Function. For more information about Functions triggers and bindings, see:

- [Azure Functions triggers and bindings concepts](#)
- [Execute an Azure Function with triggers](#)
- [Chain Azure Functions together using input and output bindings](#)

Create the Functions application

Functions follow the single responsibility principle: do only one thing. For more information about Functions development, see:

- [Azure Functions developers guide](#)
- [Create serverless applications](#)
- [Strategies for testing your code in Azure Functions](#)
- [Functions best practices](#)

Use Durable Functions for stateful workflows

Durable Functions in Azure Functions let you define stateful workflows in a serverless environment by writing *orchestrator functions*, and stateful entities by writing *entity functions*. Durable Functions manage state, checkpoints, and restarts, allowing you to focus on business logic. For more information, see [What are Durable Functions](#).

Understand and address cold starts

If the number of serverless host instances scales down to zero, the next request has the added latency of restarting the Function app, called a *cold start*. To minimize the performance impact of cold starts, reduce dependencies that the Functions app needs to load on startup, and use as few large, synchronous calls and operations as possible. For more information about autoscaling and cold starts, see [Serverless Functions operations](#).

Manage application secrets

For security, don't store credentials in application code. To use Azure Key Vault with Azure Functions to store and retrieve keys and credentials, see [Use Key Vault references for App Service and Azure Functions](#).

For more information about serverless Functions application security, see [Serverless Functions security](#).

Deployment

To prepare serverless Functions application for production, make sure you can:

- Fulfill application resource requirements.
- Monitor all aspects of the application.
- Diagnose and troubleshoot application issues.
- Deploy new application versions without affecting production systems.

Define deployment technology

Decide on deployment technology, and organize scheduled releases. For more information about how Functions app deployment enables reliable, zero-downtime upgrades, see [Deployment technologies in Azure Functions](#).

Avoid using too many resource connections

Functions in a Functions app share resources, including connections to HTTPS, databases, and services such as Azure Storage. When many Functions are running concurrently, it's possible to run out of available connections. For more information, see [Manage connections in Azure Functions](#).

Configure logging, alerting, and application monitoring

Application Insights in Azure Monitor collects log, performance, and error data. Application Insights automatically detects performance anomalies, and includes powerful analytics tools to help diagnose issues and understand function usage.

For more information about application monitoring and logging, see:

- [Monitor Azure Functions](#)
- [Monitoring Azure Functions with Azure Monitor Logs](#)

- [Application Insights for Azure Functions supported features](#)

Diagnose and troubleshoot issues

Learn how to effectively use diagnostics for troubleshooting in proactive and problem-first scenarios. For more information, see:

- [Keep your Azure App Service and Azure Functions apps healthy and happy](#)
- [Troubleshoot error: "Azure Functions Runtime is unreachable"](#)

Deploy applications using an automated pipeline and DevOps

Full automation of all steps from code commit to production deployment lets teams focus on building code, and removes the overhead and potential human error of manual steps. Deploying new code is quicker and less risky, helping teams become more agile, more productive, and more confident about their code.

For more information about DevOps and continuous deployment (CD), see:

- [Continuous deployment for Azure Functions](#)
- [Continuous delivery by using Azure DevOps](#)
- [Continuous delivery by using GitHub Action](#)

Optimization

Once the application is in production, prepare for scaling and implement site reliability engineering (SRE).

Ensure optimal scalability

For information about factors that impact Functions app scalability, see:

- [Scalability best practices](#)
- [Performance and scale in Durable Functions](#)

Implement SRE practices

Site Reliability Engineering (SRE) is a proven approach to maintaining crucial system and application reliability, while iterating at the speed the marketplace demands. For more information, see:

- [Introduction to Site Reliability Engineering \(SRE\)](#)
- [DevOps at Microsoft: Game streaming SRE](#)

Next steps

For hands-on serverless Functions app development and deployment walkthroughs, see:

- [Serverless Functions code walkthrough](#)
- [CI/CD for a serverless frontend](#)

For an engineering playbook to help teams and customers successfully implement serverless Functions projects, see the [Code-With Customer/Partner Engineering Playbook](#).

Monitor serverless event processing

3/10/2022 • 8 minutes to read • [Edit Online](#)

This article provides guidance on monitoring [serverless](#) event-driven architectures.

Monitoring provides insight into the behavior and health of your systems. It helps you build a holistic view of the environment, retrieve historic trends, correlate diverse factors, and measure changes in performance, consumption, or error rate. You can use monitoring to define alerts when conditions occur that could impact the quality of your service, or when conditions of particular interest to your specific environment arise.

This article demonstrates using [Azure Monitor](#) to monitor a serverless application built using [Event Hubs](#) and [Azure Functions](#). It discusses useful metrics to monitor, describes how to integrate with [Application Insights](#) and capture custom metrics, and provides code samples.

Assumptions

This article assumes you have an architecture like the one described in the [Serverless event processing reference architecture](#). Basically:

- Events arrive at Azure Event Hubs.
- A Function App is triggered to handle the event.
- Azure Monitor is available for use with your architecture.

Metrics from Azure Monitor

First we need to decide which metrics will be needed before we can begin to formulate useful insights about the architecture. Each resource performs different tasks, and in turn generates different metrics.

These metrics from Event Hub will be of interest to capture useful insights:

- Incoming requests
- Outgoing requests
- Throttled requests
- Successful requests
- Incoming messages
- Outgoing messages
- Captured messages
- Incoming bytes
- Outgoing bytes
- Captured bytes
- User errors

Similarly, these metrics from Azure Functions will be of interest to capture useful insights:

- Function execution count
- Connections
- Data in
- Data out
- HTTP server errors
- Requests

- Requests in application queue
- Response time

Using diagnostics logging to capture insights

When analyzed together, the above metrics can be used to formulate and capture the following insights:

- Rate of requests processed by Event Hubs
- Rate of requests processed by Azure Functions
- Total Event Hub throughput
- User errors
- Duration of Azure Functions
- End-to-end latency
- Latency at each stage
- Number of messages lost
- Number of messages processed more than once

To ensure that Event Hubs captures the necessary metrics, we must first enable diagnostic logs (which are disabled by default). We must then select which logs are desired and configure the correct Log Analytics workspace as the destination for them.

The log and metric categories that we are interested in are:

- OperationalLogs
- AutoScaleLogs
- KafkaCoordinatorLogs (*for Apache Kafka workloads*)
- KafkaUserErrorLogs (*for Apache Kafka workloads*)
- EventHubVNetConnectionEvent
- AllMetrics

Azure documentation provides instructions on how to [Set up diagnostic logs for an Azure event hub](#). The following screenshot shows an example *Diagnostic setting* configuration panel with the correct log and metric categories selected, and a Log Analytics workspace set as the destination. (If an external system is being used to analyze the logs, the option to *Stream to an event hub* can be used instead.)

Diagnostic setting



[Save](#) [Discard](#) [Delete](#) [Provide feedback](#)

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name **eh-log-analytics**

Category details

log

- ArchiveLogs
- OperationalLogs
- AutoScaleLogs
- KafkaCoordinatorLogs
- KafkaUserErrorLogs
- EventHubVNetConnectionEvent
- CustomerManagedKeyUserLogs

metric

- AllMetrics

Destination details

Send to Log Analytics workspace

Subscription

[REDACTED]

Log Analytics workspace

samltbkend-loganalytics (southcentralus)

Archive to a storage account

Stream to an event hub

NOTE

In order to utilize log diagnostics to capture insights, you should create event hubs in different namespaces. This is because of a constraint in Azure.

The Event Hubs set in a given Event Hubs namespace is represented in Azure Monitor metrics under a dimension called `EntityName`. In the Azure portal, data for a specific event hub normally can be viewed on that instance of Azure Monitor. But when the metrics data is routed to the log diagnostics, there is currently no way to view data per event hub by filtering on the `EntityName` dimension.

As a workaround, creating event hubs in different namespaces helps make it possible to locate metrics for a specific hub.

Using Application Insights

You can enable Application Insights to capture metrics and custom telemetry from Azure Functions. This allows you to define analytics that suit your own purposes, providing another way to get important insights for the serverless event processing scenario.

This screenshot shows an example listing of custom metrics and telemetry within Application Insights:

timestamp [UTC]	message	severityLevel	itemType	customDimensions
> 11/10/2020, 2:43:28.754 AM	Executing 'EventHubStandardTriggeredFunction_AsSingleEvent' (Re...	1	trace	{"prop_{OriginalFormat}": "Execut...
> 11/10/2020, 2:43:28.754 AM	Trigger Details: PartitionId: 14, Offset: 25800738240, EnqueueTimeUtc...	1	trace	{"prop_{OriginalFormat}": "Trigg...
> 11/10/2020, 2:43:28.754 AM	Executing 'EventHubStandardTriggeredFunction_AsSingleEvent' (Re...	1	trace	{"prop_{OriginalFormat}": "Execut...
> 11/10/2020, 2:43:28.754 AM	Trigger Details: PartitionId: 14, Offset: 25800738240, EnqueueTimeUtc...	1	trace	{"prop_{OriginalFormat}": "Trigg...
> 11/10/2020, 2:43:28.753 AM	Executing 'EventHubStandardTriggeredFunction_AsSingleEvent' (Re...	1	trace	{"prop_{OriginalFormat}": "Execut...
> 11/10/2020, 2:43:28.716 AM	Executed 'EventHubStandardTriggeredFunction_AsSingleEvent' (Succ...	1	trace	{"prop_{OriginalFormat}": "Execut...
> 11/10/2020, 2:43:28.716 AM	Read Payload from message: {"Body":123,34,107,101,121,34,58,34,97,...	1	trace	{"prop_{OriginalFormat}": "Read...
> 11/10/2020, 2:43:28.715 AM	Executed 'EventHubStandardTriggeredFunction_AsSingleEvent' (Succ...	1	trace	{"prop_{OriginalFormat}": "Execut...
> 11/10/2020, 2:43:28.715 AM	Read Payload from message: {"Body":123,34,107,101,121,34,58,34,53,...	1	trace	{"prop_{OriginalFormat}": "Read...

Default custom metrics

In Application Insights, custom metrics for Azure Functions are stored in the `customMetrics` table. It includes the following values, spanned over a timeline for different function instances:

- `AvgDurationMs`
- `MaxDurationMs`
- `MinDurationMs`
- `Successes`
- `Failures`
- `SuccessRate`
- `Count`

These metrics can be used to efficiently calculate the aggregated averages across the multiple function instances that are invoked in a run.

This screenshot shows what these default custom metrics look like when viewed in Application Insights:

timestamp [UTC]	name	value	valueCount	valueSum	valueMin
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent Count	1,000	1	1,000	1,000
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent AvgDurationMs	84.757	1	84.757	84.757
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent MaxDurationMs	276.234	1	276.234	276.234
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent MinDurationMs	11.744	1	11.744	11.744
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent Successes	1,000	1	1,000	1,000
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent Failures	0	1	0	0
11/10/2020, 2:36:38.242 AM	EventHubStandardTriggeredFunction_AsSingleEvent SuccessRate	100	1	100	100

Custom messages

Custom messages logged in the Azure Function code (using the `ILogger`) are obtained from the Application Insights `traces` table.

The `traces` table has the following important properties (among others):

- `timestamp`
- `cloud_RoleInstance`
- `operation_Id`
- `operation_Name`
- `message`

Here is an example of what a custom message might look like in the Application Insights interface:

▼	11/10/2020, 2:47:27.960 AM	Read Payload from message: {"Body":123,34,107,101,121,34,58,34,52,98,100,54,57,48,100,52,45,101,98,54,54,45,52,...}	1
	timestamp [UTC]	2020-11-10T02:47:27.9609674Z	
	message	Read Payload from message: {"Body":123,34,107,101,121,34,58,34,52,98,100,54,57,48,100,52,45,101,98,54,54,45,52,54,50,101,4}	
	severityLevel	1	
	itemType	trace	
➤	customDimensions	{"prop_{OriginalFormat}":"Read Payload from message: \\"Body\\":123,34,107,101,121,34,58,34,52,98,100,54,57,48,100,52,45,101,98,54,54,45,52,54,50,101,4"} {"prop_OriginalFormat": "Read Payload from message: \\"Body\\":123,34,107,101,121,34,58,34,52,98,100,54,57,48,100,52,45,101,98,54,54,45,52,54,50,101,4"}	
	operation_Name	EventHubStandardTriggeredFunction_AsSingleEvent	
	operation_Id	6b66dca58934a04cabdb26acd4e29a0c	
	operation_ParentId	f67b02405e1cf146	
	client_Type	PC	
	client_IP	0.0.0.0	
	cloud_RoleName	samltbkendwinyfx-dotnet	
	cloud_RoleInstance	b637eff58cea882e5c19a73e4e44cca8088f890a8b41921aa654b05c5e871721	
	appId	0e0902f4-e3fd-4624-9dec-64bb06751b50	
	appName	samltbkendwinyfx	
	iKey	47171669-fd25-42a5-b7b3-552d9168743d	
	sdkVersion	azurefunctions: 3.0.14916.0	
	itemId	12271dcd-22ff-11eb-b726-fff2d030959d	
	itemCount	1	

If the incoming Event Hub message or `EventData[]` is logged as a part of this custom `ILogger` message, then that is also made available in Application Insights. This can be very useful.

For our serverless event processing scenario, we log the JSON serialized message body that's received from the event hub. This allows us to capture the raw byte array, along with `SystemProperties` like `x-opt-sequence-number`, `x-opt-offset`, and `x-opt-enqueued-time`. To determine when each message was received by the Event Hub, the `x-opt-enqueued-time` property is used.

Sample query:

```
traces
| where timestamp between(min_t .. max_t)
| where message contains "Body"
| extend m = parse_json(message)
| project timestamp = todatetime(m.SystemProperties["x-opt-enqueued-time"])
```

The sample query would return a message similar to the following example result, which gets logged by default in Application Insights. The properties of the `Trigger Details` can be used to locate and capture additional insights around messages received per `PartitionId`, `Offset`, and `SequenceNumber`.

Example result of the sample query:

```
"message": Trigger Details: PartitionId: 26, Offset: 17194119200, EnqueueTimeUtc: 2020-11-03T02:14:01.7740000Z, SequenceNumber: 843572, Count: 10,
```

WARNING

The library for Azure Java Functions currently has an issue that prevents access to the `PartitionID` and the `PartitionContext` when using `EventHubTrigger`. Learn more in this [GitHub issue report](#).

Tracking message flow using a transaction ID with Application Insights

In Application Insights, we can view all the telemetry related to a particular transaction by doing a Transaction

search query on the transaction's `Operation Id` value. This can be especially useful for capturing the percentile values of average times for messages as the transaction moves through the event stream pipeline.

The following screenshot shows an example Transaction search in the Application Insights interface. The desired `Operation ID` is entered in the query field, identified with a magnifying glass icon (and shown here outlined in a red box). At the bottom of the main pane, the `Results` tab shows matching events in sequential order. In each event entry, the `operation_id` value is highlighted in dark blue for easy verification.

Samltbkndwinyfx | Transaction search

Application Insights

Search (Ctrl+)

Refresh Reset View in Logs Feedback Help Open Classic Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Investigate

Application map

Smart Detection

Live Metrics

Transaction search

Availability

Failures

Performance

Troubleshooting guides (prev...)

Monitoring

Alerts

Metrics

Logs

Workbooks

Usage

Users

Sessions

Events

Funnels

Local time: 10/8 11:58 PM - 11/10 10:58 PM Event types: All selected

1c89d7073a00eabbcc8f2e65670e46

5 total results between 10/08/2020, 11:58:40 PM and 11/10/2020, 10:58:40 PM

6

4

2

0

Fri 9 Sun 11 Tue 13 Thu 15 Sat 17 Mon 19 Wed 21 Fri 23 Sun 25 Tue 27 Thu 29 Sat 31 Sun 1 Tue 3 Thu 5 Sat 7 Mon 9

11:58 PM

Availability Request Exception Page View Trace Custom Event Dependency

0 1 0 0 4 0 0

Results Grouped results. Loading... 0 loaded

Sort by time Newest first Oldest first

11/9/2020, 7:02:01 PM - TRACE Executed 'EventHubStandardTriggeredFunction_AcSingleEvent' (Succeeded, Id-b6d63bdd-2d20-461f-94f9-4fd46567ce72, Duration=28ms) Severity Level Information OperationId: 1c89d7073a00eabbcc8f2e65670e46

11/9/2020, 7:02:01 PM - TRACE Read Payload from message : {"Body": [123, 34, 107, 101, 121, 34, 58, 34, 56, 57, 102, 97, 56, 102, 101, 45, 55, 56, 98, 57, 45, 52, 98, 101, 50, 45, 97, 97, 53, 100, 45, 51, 56, 101, 51, 101, 57, 98, 97, 52, 34] Severity Level Information OperationId: 1c89d7073a00eabbcc8f2e65670e46

11/9/2020, 7:02:01 PM - TRACE Trigger Details: PartitionId: 25, Offset: 208684705888, EnqueueTimeUtc: 2020-11-10T02:43:11.658000Z, SequenceNumber: 1447822, Count: 10 Severity Level Information OperationId: 1c89d7073a00eabbcc8f2e65670e46

11/9/2020, 7:02:01 PM - TRACE Executing 'EventHubStandardTriggeredFunction_AcSingleEvent' (Reason='(null)', Id-b6d63bdd-2d20-461f-94f9-4fd46567ce72) Severity Level Information OperationId: 1c89d7073a00eabbcc8f2e65670e46

11/9/2020, 7:02:01 PM - REQUEST EventHubStandardTriggeredFunction_AcSingleEvent Request URL: <empty> Response code: 0 Response time: 28.44 ms OperationId: 1c89d7073a00eabbcc8f2e65670e46

A query generated for a specific operation ID will look like the following. Note that the `operation_id` GUID is specified in the third line's `where * has` clause. This example further narrows the query between two different `datetimes`.

```
union isfuzzy=true availabilityResults, requests, exceptions, pageViews, traces, customEvents, dependencies  
| where timestamp > datetime("2020-10-09T06:58:40.024Z") and timestamp < datetime("2020-11-  
11T06:58:40.024Z")  
| where * has "1c8c9d7073a00e4bbdcc8f2e6570e46"  
| order by timestamp desc  
| take 100
```

Here is a screenshot of what the query and its matching results might look like in the Application Insights interface:

```
1 union isfuzzy=true availabilityResults, requests, exceptions, pageViews, traces, customEvents, dependencies
2 | where timestamp > datetime("2020-10-09T06:58:40.024Z") and timestamp < datetime("2020-11-11T06:58:40.024Z")
3 | where * has "1c8cd7073aa0e4bbdcfc8f2e65670e46"
4 | order by timestamp desc
5 | take 100
```

Results	Chart	Columns	Display time (UTC+0:00)	Group columns			
Completed					🕒 00:06.2	5 records	✖
timestamp [UTC]	↑	⬇ id	⬇ name	⬇ success	⬇ message	⬇ duration	⬇ perform
➤ 11/10/2020, 3:02:01.037 AM		66c6044ed1be1040	EventHubStandardTriggeredFunction_AsSingleEvent	True		28.436	<250ms
➤ 11/10/2020, 3:02:01.037 AM					Executing 'EventHubStandardTriggeredFunction_AsSingleEvent' (Re...		
➤ 11/10/2020, 3:02:01.037 AM					Trigger Details: PartitionId: 25, Offset: 30084705888, EnqueueTimeUtc:...		
➤ 11/10/2020, 3:02:01.065 AM					Read Payload from message: {"Body":{123,34,107,101,121,34,58,34,56,...}		
➤ 11/10/2020, 3:02:01.065 AM					Executed 'EventHubStandardTriggeredFunction_AsSingleEvent' (Succ...		

Capture custom metrics from Azure Functions

.NET functions

Structured logging is used in the .NET Azure functions for capturing custom dimensions in the Application Insights traces table. These custom dimensions can then be used for querying data.

As an example, here is the log statement in the .NET `TransformingFunction`:

```
log.LogInformation("TransformingFunction: Processed sensorDataJson={sensorDataJson}, " +
    "partitionId={partitionId}, offset={offset} at {enqueuedTimeUtc}, " +
    "inputEH_enqueuedTime={inputEH_enqueuedTime}, processedTime={processedTime}, " +
    "transformingLatencyInMs={transformingLatencyInMs}, processingLatencyInMs={processingLatencyInMs}",
    sensorDataJson,
    partitionId,
    offset,
    enqueuedTimeUtc,
    inputEH_enqueuedTime,
    processedTime,
    transformingLatency,
    processingLatency);
```

The resulting logs created on Application Insights contain the above parameters as custom dimensions, as shown in this screenshot:

▼ 2/5/2021, 6:16:13.763 AM	TransformingFunction: Processed sensorDataJson={"Value":"eyJsb2N... 1	trace	{"prop_{OriginalFormat}"}
timestamp [UTC]	2021-02-05T06:16:13.7633341Z		
message	TransformingFunction: Processed sensorDataJson={"Value":"eyJsb2NhdGlvbil6IINlYXR0bGUiLCJyZWFlkaW5nljozMS4yLCJjYX		
severityLevel	1		
itemType	trace		
▼ customDimensions	{"prop_{OriginalFormat}": "TransformingFunction: Processed sensorDataJson={sensorDataJson}, partitionId={partitionId}, of		
Category	Function.TransformingFunction.User		
HostInstanceId	63f8de3b-42c9-4cf8-8585-c62ff9d7dfb8		
InvocationId	b24e2f69-90e9-4c4d-a20f-70ea23f3a051		
LogLevel	Information		
ProcessId	10088		
prop_Links	System.Collections.Generic.List`1[System.Diagnostics.Activity]		
prop_enqueuedTimeUtc	2021-02-05T06:16:13.6590000Z		
prop_inputEH_enqueuedTime	2021-02-05T06:16:13.6390000Z		
prop_offset	11846672		
prop_partitionId	22		
prop_processedTime	2021-02-05T06:16:13.7630000Z		
prop_processingLatency	00:00:00.1242571		
▶ prop_sensorDataJson	{"Value":"eyJsb2NhdGlvbil6IINlYXR0bGUiLCJyZWFlkaW5nljozMS4yLCJjYXRIZ29yeSI6lnRlbXBlcmtF0dXJlIn		
prop_transformingLatency	00:00:00.1042571		
prop_{OriginalFormat}	TransformingFunction: Processed sensorDataJson={sensorDataJson}, partitionId={partitionId}, offset={o		

These logs can be queried as follows:

```
traces
| where timestamp between(min_t .. max_t)
// Function name should be of the function consuming from the Event Hub of interest
| where operation_Name == "{Function_Name}"
| where message has "{Function_Name}: Processed"
| project timestamp = todatetime(customDimensions.prop_enqueuedTimeUtc)
```

NOTE

In order to make sure we do not affect performance in these tests, we have turned on the sampling settings of Azure Function logs for Application Insights using the `host.json` file as shown below. This means that all statistics captured from logging are considered to be average values and not actual counts.

host.json example:

```
"logging": {  
    "applicationInsights": {  
        "samplingExcludedTypes": "Request",  
        "samplingSettings": {  
            "isEnabled": true  
        }  
    }  
}
```

Java functions

Currently, structured logging isn't supported in Java Azure functions for capturing custom dimensions in the Application Insights traces table.

As an example, here is the log statement in the Java `TransformingFunction`:

```
LoggingUtilities.logSuccessInfo(  
    context.getLogger(),  
    "TransformingFunction",  
    "SuccessInfo",  
    offset,  
    processedTimeString,  
    dateFormatter.format(enqueuedTime),  
    transformingLatency  
) ;
```

The resulting logs created on Application Insights contain the above parameters in the message as shown below:

2/25/2021, 11:44:05.958 PM	{"functionName": "TransformingFunction", "logType": "SuccessInfo", "..."}	1	trace	{"HostInstanceId": "bd6b9e0b-4e58-4b21-b7a1-e906feae502", "InvocationId": "b1ac3952-345a-4b83-bcaf-9fe4e5f8984d", "LogLevel": "Information", "Category": "Function.TransformingFunction.User", "ProcessId": "5832"}	Transfo
timestamp [UTC]	2021-02-25T23:44:05.958379Z				
message	{"functionName": "TransformingFunction", "logType": "SuccessInfo", "offset": "17189587000", "processedTime": "2021-02-25T23:44:05.930251Z", "enqueuedTime": "2021-02-25T23:19:42.018Z", "partitionKey": "null", "latency": "2393233", "logType": "SuccessInfo", "offset": "17189587000", "partitionKey": "null", "processedTime": "2021-02-25T23:44:05.930251Z", "enqueuedTime": "2021-02-25T23:19:42.018Z", "latency": "2393233", "partitionKey": "null"}				
enqueuedTime	2021-02-25T23:19:42.018Z				
functionName	TransformingFunction				
latency	2393233				
logType	SuccessInfo				
offset	17189587000				
partitionKey	null				
processedTime	2021-02-25T23:44:05.930251Z				
severityLevel	1				
itemType	trace				
customDimensions	{"HostInstanceId": "bd6b9e0b-4e58-4b21-b7a1-e906feae502", "InvocationId": "b1ac3952-345a-4b83-bcaf-9fe4e5f8984d", "LogLevel": "Information", "Category": "Function.TransformingFunction.User", "ProcessId": "5832"}				

These logs can be queried as follows:

```
traces  
| where timestamp between(min_t .. max_t)  
// Function name should be of the function consuming from the Event Hub of interest  
| where operation_Name in ("{{Function name}}") and message contains "SuccessInfo"  
| project timestamp = todatetime(tostring(parse_json(message).enqueuedTime))
```

NOTE

In order to make sure we do not affect performance in these tests, we have turned on the sampling settings of Azure Function logs for Application Insights using the `host.json` file as shown below. This means that all statistics captured from logging are considered to be average values and not actual counts.

host.json example:

```
"logging": {  
    "applicationInsights": {  
        "samplingExcludedTypes": "Request",  
        "samplingSettings": {  
            "isEnabled": true  
        }  
    }  
}
```

Related resources

- [Serverless event processing](#) is a reference architecture detailing a typical architecture of this type, with code samples and discussion of important considerations.
- [De-batching and filtering in serverless event processing with Event Hubs](#) describes in more detail how these portions of the reference architecture work.
- [Private link scenario in event stream processing](#) is a solution idea for implementing a similar architecture in a virtual network (VNet) with private endpoints, in order to enhance security.
- [Azure Kubernetes in event stream processing](#) describes a variation of a serverless event-driven architecture running on Azure Kubernetes with KEDA scaler.

Serverless Functions app operations

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article describes Azure operations considerations for serverless Functions applications. To support Functions apps, operations personnel need to:

- Understand and implement hosting configurations.
- Future-proof scalability by automating infrastructure provisioning.
- Maintain business continuity by meeting availability and disaster recovery requirements.

Planning

To plan operations, understand your workloads and their requirements, then design and configure the best options for the requirements.

Choose a hosting option

The Azure Functions Runtime provides flexibility in hosting. Use the [hosting plan comparison table](#) to determine the best choice for your requirements.

- Azure Functions hosting plans

Each Azure Functions project deploys and runs in its own Functions app, which is the unit of scale and cost. The three hosting plans available for Azure Functions are the Consumption plan, Premium plan, and Dedicated (App Service) plan. The hosting plan determines scaling behavior, available resources, and support for advanced features like virtual network connectivity.

- Azure Kubernetes Service (AKS)

Kubernetes-based Functions provides the Functions Runtime in a Docker container with event-driven scaling through Kubernetes-based Event Driven Autoscaling (KEDA).

For more information about hosting plans, see:

- [Azure Functions scale and hosting](#)
- [Consumption plan](#)
- [Premium plan](#)
- [Dedicated \(App Service\) plan](#)
- [Azure Functions on Kubernetes with KEDA](#)
- [Azure subscription and service limits, quotas, and constraints](#)

Understand scaling

The serverless Consumption and Premium hosting plans *scale* automatically, adding and removing Azure Functions host instances based on the number of incoming events. Scaling can vary on several dimensions, and behave differently based on plan, trigger, and code language.

For more information about scaling, see:

- [Understand scaling behaviors](#)
- [Scalability best practices](#)

Understand and address cold starts

If the number of host instances scales down to zero, the next request has the added latency of restarting the

Function app, called a *cold start*. [Cold start](#) is a large discussion point for serverless architectures, and a point of ambiguity for Azure Functions.

The Premium hosting plan prevents cold starts by keeping some instances warm. Reducing dependencies and using asynchronous operations in the Functions app also minimizes the impact of cold starts. However, availability requirements may require running the app in a Dedicated hosting plan with *Always on* enabled. The Dedicated plan uses dedicated virtual machines (VMs), so is not serverless.

For more information about cold start, see [Understanding serverless cold start](#).

Identify storage considerations

Every Azure Functions app relies on Azure Storage for operations such as managing triggers and logging function executions. When creating a Functions app, you must create or link to a general-purpose Azure Storage account that supports Blob, Queue, and Table storage. For more information, see [Storage considerations for Azure Functions](#).

Identify network design considerations

Networking options let the Functions app restrict access, or access resources without using internet-routable addresses. The hosting plans offer different levels of network isolation. Choose the option that best meets your network isolation requirements. For more information, see [Azure Functions networking options](#).

Production

To prepare the application for production, make sure you can easily redeploy the hosting plan, and apply scale-out rules.

Automate hosting plan provisioning

With infrastructure as code, you can automate infrastructure provisioning. Automatic provisioning provides more resiliency during disasters, and more agility to quickly redeploy the infrastructure as needed.

For more information on automated provisioning, see:

- [Automate resource deployment for your function app in Azure Functions](#)
- [Terraform - Manages a Function App](#)

Configure scale out options

Autoscale provides the right amount of running resources to handle application load. Autoscale adds resources to handle increases in load, and saves money by removing resources that are idle.

For more information about autoscale options, see:

- [Premium Plan settings](#)
- [App Service Plan settings](#)

Optimization

When the application is in production, make sure that:

- The hosting plan can scale to meet application demands.
- There's a plan for business continuity, availability, and disaster recovery.
- You can monitor hosting and application health and receive alerts.

Implement availability requirements

Azure Functions run in a specific region. To get higher availability, you can deploy the same Functions app to multiple regions. In multiple regions, Functions can run in the *active-active* or *active-passive* availability pattern.

For more information about Azure Functions availability and disaster recovery, see:

- [Azure Functions geo-disaster recovery](#)
- [Disaster recovery and geo-distribution in Azure Durable Functions](#)

Monitoring logging, application monitoring, and alerting

Application Insights and logs in Azure Monitor automatically collect log, performance, and error data and detect performance anomalies. Azure Monitor includes powerful analytics tools to help diagnose issues and understand function use. Application Insights help you continuously improve performance and usability.

For more information about monitoring and analyzing Azure Functions performance, see:

- [Monitor Azure Functions](#)
- [Monitor Azure Functions with Azure Monitor logs](#)
- [Application Insights for Azure Functions supported features](#)

Next steps

- [Serverless application development and deployment](#)
- [Azure Functions app security](#)

Serverless Functions security

3/10/2022 • 5 minutes to read • [Edit Online](#)

This article describes Azure services and activities security personnel can implement for serverless Functions. These guidelines and resources help develop secure code and deploy secure applications to the cloud.

Planning

The primary goals of a secure serverless Azure Functions application environment are to protect running applications, quickly identify and address security issues, and prevent future similar issues.

The [OWASP Serverless Top 10](#) describes the most common serverless application security vulnerabilities, and provides basic techniques to identify and protect against them.

In many ways, planning for secure development, deployment, and operation of serverless functions is much the same as for any web-based or cloud hosted application. Azure App Service provides the hosting infrastructure for your function apps. [Securing Azure Functions](#) article provides security strategies for running your function code, and how App Service can help you secure your functions.

For more information about Azure security, best practices, and shared responsibilities, see:

- [Security in Azure App Service](#)
- [Built-in security controls](#)
- [Secure development best practices on Azure](#).
- [Security best practices for Azure solutions \(PDF report\)](#)
- [Shared responsibilities for cloud computing \(PDF report\)](#)

Deployment

To prepare serverless Functions applications for production, security personnel should:

- Conduct regular code reviews to identify code and library vulnerabilities.
- Define resource permissions that Functions needs to execute.
- Configure network security rules for inbound and outbound communication.
- Identify and classify sensitive data access.

The [Azure Security Baseline for Azure Functions](#) article contains more recommendations that will help you improve the security posture of your deployment.

Keep code secure

Find security vulnerabilities and errors in code and manage security vulnerabilities in projects and dependencies.

For more information, see:

- [GitHub - Finding security vulnerabilities and errors in your code](#)
- [GitHub - Managing security vulnerabilities in your project](#)
- [GitHub - Managing vulnerabilities in your project's dependencies](#)

Perform input validation

Different event sources like Blob storage, Cosmos DB NoSQL databases, event hubs, queues, or Graph events can trigger serverless Functions. Injections aren't strictly limited to inputs coming directly from the API calls.

Functions may consume other input from the possible event sources.

In general, don't trust input or make any assumptions about its validity. Always use safe APIs that sanitize or validate the input. If possible, use APIs that bind or parameterize variables, like using prepared statements for SQL queries.

For more information, see:

- [Azure Functions Input Validation with FluentValidation](#)
- [Security Frame: Input Validation Mitigations](#)
- [HTTP Trigger Function Request Validation](#)
- [How to validate request for Azure Functions](#)

Secure HTTP endpoints for development, testing, and production

Azure Functions lets you use keys to make it harder to access your HTTP function endpoints. To fully secure your function endpoints in production, consider implementing one of the following Function app-level security options:

- Turn on App Service authentication and authorization for your Functions app. See [Authorization keys](#).
- Use Azure API Management (APIM) to authenticate requests. See [Import an Azure Function App as an API in Azure API Management](#).
- Deploy your Functions app to an Azure App Service Environment (ASE).
- Use an App Service Plan that restricts access, and implement Azure Front Door + WAF to handle your incoming requests. See [Create a Front Door for a highly available global web application](#).

For more information, see [Secure an HTTP endpoint in production](#).

Set up Azure role-based access control (Azure RBAC)

Azure role-based access control (Azure RBAC) has several Azure built-in roles that you can assign to users, groups, service principals, and managed identities to control access to Azure resources. If the built-in roles don't meet your organization's needs, you can create your own Azure custom roles.

Review each Functions app before deployment to identify excessive permissions. Carefully examine functions to apply "least privilege" permissions, giving each function only what it needs to successfully execute.

Use Azure RBAC to assign permissions to users, groups, and applications at a certain scope. The scope of a role assignment can be a subscription, a resource group, or a single resource. Avoid using wildcards whenever possible.

For more information about Azure RBAC, see:

- [What is Azure role-based access control \(Azure RBAC\)?](#)
- [Azure built-in roles](#)
- [Azure custom roles](#)

Use managed identities and key vaults

A common challenge when building cloud applications is how to manage credentials for authenticating to cloud services in your code. Credentials should never appear in application code, developer workstations, or source control. Instead, use a key vault to store and retrieve keys and credentials. Azure Key Vault provides a way to securely store credentials, secrets, and other keys. The code authenticates to Key Vault to retrieve the credentials.

For more information, see [Use Key Vault references for App Service and Azure Functions](#).

Managed identities let Functions apps access resources like key vaults and storage accounts without requiring specific access keys or connection strings. A full audit trail in the logs displays which identities execute requests to resources. Use Azure RBAC and managed identities to granularly control exactly what resources Azure

Functions applications can access.

For more information, see:

- [What are managed identities for Azure resources?](#)
- [How to use managed identities for App Service and Azure Functions](#)

Use shared access signature (SAS) tokens to limit access to resources

A *shared access signature (SAS)* provides secure delegated access to resources in your storage account, without compromising the security of your data. With a SAS, you have granular control over how a client can access your data. You can control what resources the client may access, what permissions they have on those resources, and how long the SAS is valid, among other parameters.

For more information, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

Secure Blob storage

Identify and classify sensitive data, and minimize sensitive data storage to only what is necessary. For sensitive data storage, add multi-factor authentication and data encryption in transit and at rest. Grant limited access to Azure Storage resources using SAS tokens.

For more information, see [Security recommendations for Blob storage](#).

Optimization

Once an application is in production, security personnel can help optimize workflow and prepare for scaling.

Use Microsoft Defender for Cloud and apply security recommendations

Microsoft Defender for Cloud is a security scanning solution for your application that identifies potential security vulnerabilities and creates recommendations. The recommendations guide you to configure needed controls to harden and protect your resources.

For more information, see:

- [Protect your applications with Microsoft Defender for Cloud](#)
- [Defender for Cloud app recommendations](#)

Enforce application governance policies

Apply centralized, consistent enforcements and safeguards to your application at scale. For more information, see [Azure Policy built-in policy definitions](#).

Next steps

- [Serverless application development and deployment](#)
- [Azure Functions app operations](#)

Resilient Event Hubs and Functions design

3/10/2022 • 11 minutes to read • [Edit Online](#)

Error handling, designing for idempotency and managing retry behavior are a few of the critical measures you can take to ensure Event Hubs triggered functions are resilient and capable of handling large volumes of data. This article covers these crucial concepts and makes recommendations for serverless event-streaming solutions.

Azure provides three main messaging services that can be used with Azure Functions to support a wide range of unique, event-driven scenarios. Because of its partitioned consumer model and ability to ingest data at a high rate, Azure Event Hubs is commonly used for event streaming and big data scenarios. For a detailed comparison of Azure messaging services, see [Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus](#).

Streaming benefits and challenges

Understanding the benefits and drawbacks of streams helps you appreciate how a service like [Event Hubs](#) operates. You also need this context when making impactful architectural decisions, troubleshooting issues, and optimizing for performance. Consider the following key concepts about solutions featuring both Event Hubs and Functions:

- **Streams are not queues:** Event Hubs, Kafka, and other similar offerings that are built on the partitioned consumer model don't intrinsically support some of the principal features in a message broker like [Service Bus](#). Perhaps the biggest indicator of this is the fact that reads are **non-destructive**. This means that the data that is read by the Functions host isn't deleted afterwards. Instead, messages are immutable and remain for other consumers to read, including potentially the same customer reading it again. For this reason, solutions that implement patterns such as [competing consumers](#) are better suited for a traditional message broker.
- **Missing inherit dead-letter support:** A dead-letter channel is not a native feature in Event Hubs or Kafka. Often, the *concept* of dead-lettering is integrated into a streaming solution to account for data that cannot be processed. This functionality is intentionally not an innate element in Event Hubs and is only added on the consumer side to manufacture a similar behavior or effect. If you need dead-letter support, you should potentially review your choice of streaming message service.
- **A unit of work is a partition:** In a traditional message broker, a unit of work is a single message. In a streaming solution, a partition is often considered the unit of work. If each event in an event hub is regarded as a discrete message that requires it to be treated like an order processing operation or financial transaction, it's most likely an indication of the wrong messaging service being used.
- **No server-side filtering:** One of the reasons Event Hubs is capable of tremendous scale and throughput is due to the low overhead on the service itself. Features like server-side filtering, indexes, and cross-broker coordination aren't part of the architecture of Event Hubs. Functions are occasionally used to filter events by routing them to other Event Hubs based on the contents in the body or header. This approach is common in event streaming but comes with the caveat that each event is read and evaluated by the initial function.
- **Every reader must read all data:** Since server-side filtering is unavailable, a consumer sequentially reads all the data in a partition. This includes data that may not be relevant or could even be malformed. There are several options and even strategies that can be used to compensate for these challenges that will be covered later in this section.

These significant design decisions allow Event Hubs to do what it does best: support a significant influx of events

and provide a robust and resilient service for consumers to read from. Each consumer application is tasked with the responsibility of maintaining their own, client-side offsets or cursor to those events. The low overhead makes Event Hubs an affordable and powerful option for event streaming.

Idempotency

One of the core tenets of Azure Event Hubs is the concept of at-least once delivery. This approach ensures that events will always be delivered. It also means that events can be received more than once, even repeatedly, by consumers such as a function. For this reason, it's important that an event hub triggered function supports the [idempotent consumer](#) pattern.

Working under the assumption of at-least once delivery, especially within the context of an event-driven architecture, is a responsible approach for reliably processing events. Your function must be idempotent so that the outcome of processing the same event multiple times is the same as processing it once.

Duplicate events

There are several different scenarios that could result in duplicate events being delivered to a function:

- **Checkpointing:** If the Azure Functions host crashes, or the threshold set for the [batch checkpoint frequency](#) is not met, a checkpoint will not be created. As a result, the offset for the consumer is not advanced and the next time the function is invoked, it will resume from the last checkpoint. It is important to note that checkpointing occurs at the partition level for each consumer.
- **Duplicate events published:** There are many techniques that could alleviate the possibility of the same event being published to a stream, however, it's still the responsibility of the consumer to idempotently handle duplicates.
- **Missing acknowledgments:** In some situations, an outgoing request to a service may be successful, however, an acknowledgment (ACK) from the service is never received. This might result in the perception that the outgoing call failed and initiate a series of retries or other outcomes from the function. In the end, duplicate events could be published, or a checkpoint is not created.

Deduplication techniques

Designing your functions for [identical input](#) should be the default approach taken when paired with the Event Hub trigger binding. You should consider the following techniques:

- **Looking for duplicates:** Before processing, take the necessary steps to validate that the event should be processed. In some cases, this will require an investigation to confirm that it is still valid. It could also be possible that handling the event is no longer necessary due to data freshness or logic that invalidates the event.
- **Design events for idempotency:** By providing additional information within the payload of the event, it may be possible to ensure that processing it multiple times will not have any detrimental effects. Take the example of an event that includes an amount to withdrawal from a bank account. If not handled responsibly, it is possible that it could decrement the balance of an account multiple times. However, if the same event includes the updated balance to the account, it could be used to perform an upsert operation to the bank account balance. This event-carried state transfer approach occasionally requires coordination between producers and consumers and should be used when it makes sense to participating services.

Error handling and retries

Error handling and retries are a few of the most important qualities of distributed, event-driven applications, and Functions are no exception. For event streaming solutions, the need for proper error handling support is crucial, as thousands of events can quickly turn into an equal number of errors if they are not handled correctly.

Error handling guidance

Without error handling, it can be tricky to implement retries, detect runtime exceptions, and investigate issues. Every function should have at least some level of error handling. A few recommended guidelines are:

- **Use Application Insights:** Enable and use Application Insights to log errors and monitor the health of your functions. Be mindful of the configurable sampling options for scenarios that process a high volume of events.
- **Add structured error handling:** Apply the appropriate error handling constructs for each programming language to catch, log, and detect anticipated and unhandled exceptions in your function code. For instance, use a try/catch block in C#, Java and JavaScript and take advantage of the [try and except](#) blocks in Python to handle exceptions.
- **Logging:** Catching an exception during execution provides an opportunity to log critical information that could be used to detect, reproduce, and fix issues reliably. Log the exception, not just the message, but the body, inner exception and other useful artifacts that will be helpful later.
- **Do not catch and ignore exceptions:** One of the worst things you can do is catch an exception and do nothing with it. If you catch a generic exception, log it somewhere. If you don't log errors, it's difficult to investigate bugs and reported issues.

Retries

Implementing retry logic in an event streaming architecture can be complex. Supporting cancellation tokens, retry counts and exponential back off strategies are just a few of the considerations that make it challenging. Fortunately, Functions provides [retry policies](#) that can make up for many of these tasks that you would typically code yourself.

Several important factors that must be considered when using the retry policies with the Event Hub binding, include:

- **Avoid indefinite retries:** When the [max retry count](#) setting is set to a value of -1, the function will retry indefinitely. In general, indefinite retries should be used sparingly with Functions and almost never with the Event Hub trigger binding.
- **Choose the appropriate retry strategy:** A [fixed delay](#) strategy may be optimal for scenarios that receive back pressure from other Azure services. In these cases, the delay can help avoid throttling and other limitations encountered from those services. The [exponential back off](#) strategy offers more flexibility for retry delay intervals and is commonly used when integrating with third-party services, REST endpoints, and other Azure services.
- **Keep intervals and retry counts low:** When possible, try to maintain a retry interval shorter than one minute. Also, keep the maximum number of retry attempts to a reasonably low number. These settings are especially pertinent when running in the Functions Consumption plan.
- **Circuit breaker pattern:** A transient fault error from time to time is expected and a natural use case for retries. However, if a significant number of failures or issues are occurring during the processing of the function, it may make sense to stop the function, address the issues and restart later.

An important takeaway for the retry policies in Functions is that it is a best effort feature for reprocessing events. It does not substitute the need for error handling, logging, and other important patterns that provide resiliency to your code.

Strategies for failures and corrupt data

There are several noteworthy approaches that you can use to compensate for issues that arise due to failures or bad data in an event stream. Some fundamental strategies are:

- **Stop sending and reading:** Pause the reading and writing of events to fix the underlying issue. The

benefit of this approach is that data won't be lost, and operations can resume after a fix is rolled out. This approach may require a circuit-breaker component in the architecture and possibly a notification to the affected services to achieve a pause. In some cases, stopping a function may be necessary until the issues are resolved.

- **Drop messages:** If messages aren't important or are considered non-mission critical, consider moving on and not processing them. This doesn't work for scenarios that require strong consistency such as recording moves in a chess match or finance-based transactions. Error handling inside of a function is recommended for catching and dropping messages that can't be processed.
- **Retry:** There are many situations that may warrant the reprocessing of an event. The most common scenario would be a transient error encountered when calling another service or dependency. Network errors, service limits and availability, and strong consistency are perhaps the most frequent use cases that justify reprocessing attempts.
- **Dead letter:** The idea here is to publish the event to a different event hub so that the existing flow is not interrupted. The perception is that it has been moved off the hot path and could be dealt with later or by a different process. This solution is used frequently for handling poisoned messages or events. It should be noted that each function, that is configured with a different consumer group, will still encounter the bad or corrupt data in their stream and must handle it responsibly.
- **Retry and dead letter:** The combination of numerous retry attempts before ultimately publishing to a dead letter stream once a threshold is met, is another familiar method.
- **Use a schema registry:** A schema registry can be used as a proactive tool to help improve consistency and data quality. The [Azure Schema Registry](#) can support the transition of schemas along with versioning and different compatibility modes as schemas evolve. At its core, the schema will serve as a contract between producers and consumers, which could reduce the possibility of invalid or corrupt data being published to the stream.

In the end, there isn't a perfect solution and the consequences and tradeoffs of each of the strategies needs to be thoroughly examined. Based on the requirements, using several of these techniques together may be the best approach.

Next steps

Before continuing, consider reviewing these related articles:

- [Azure Functions reliable event processing](#)
- [Designing Azure Functions for identical input](#)
- [Azure Functions error handling and retry guidance](#)

Security

Related resources

- [Monitoring serverless event processing](#) provides guidance on monitoring serverless event-driven architectures.
- [Serverless event processing](#) is a reference architecture detailing a typical architecture of this type, with code samples and discussion of important considerations.
- [De-batching and filtering in serverless event processing with Event Hubs](#) describes in more detail how these portions of the reference architecture work.

Secure Azure Functions with Event Hubs

3/10/2022 • 4 minutes to read • [Edit Online](#)

When configuring access to resources in Azure, you should apply fine-grained control over permissions to resources. Access to these resources should be based on *need to know* and *least privilege* security principles to make sure that clients can only perform the limited set of actions assigned to them.

Authorizing Access to Event Hubs

Authorizing access to Azure Event Hubs resources can be done using the following security constructs:

- **Azure Active Directory:** Azure Active Directory (Azure AD) provides role-based access control (RBAC) for granular control over a client's access to Event Hubs resources. Based on roles and permissions granted, Azure AD will authorize requests using an OAuth 2.0 access token.
- **Shared access signature:** A shared access signature (SAS) offers the ability to protect Event Hubs resources based on authorization rules. You define authorization policies by selecting one or more [policy rules](#), such as the ability to send messages, listen to messages, and manage the entities in the namespace.

Shared access signature considerations

When using a shared access signature with Azure Functions and Event Hubs, the following considerations should be reviewed:

- **Avoid the Manage right:** In addition to being able to manage the entities in an Event Hubs namespace, the Manage right includes both Send and Listen rights. Ideally, a function app should only be granted a combination of the Send and Listen rights, based on the actions they perform.
- **Don't use the default Manage rule:** Avoid using the default policy rule named `RootManageSharedAccessKey` unless it's needed by your function app, which should be an uncommon scenario. Another caveat to this default rule is that it's created at the namespace level and grants permissions to all underlying event hubs.
- **Review shared access policy scopes:** Shared access policies can be created at the namespace level and per event hub. Consider creating granular access policies that are tailored for each client to limit their range and permissions.

Managed identity

An Active Directory identity can be assigned to a managed resource in Azure such as a function app or web app. Once an identity is assigned, it has the capabilities to work with other resources that use Azure AD for authorization, much like a [service principal](#).

Function apps can be assigned a [managed identity](#) and take advantage of identity-based connections for a subset of services, including Event Hubs. Identity-based connections provide support for both the trigger and output binding extensions and must use the [Event Hubs extension 5.x and higher](#) for support.

Network

By default, Event Hubs namespaces are accessible from the internet, so long as the request comes with valid authentication and authorization. There are three options for limiting network access to Event Hubs namespaces:

- [Allow access from specific IP addresses](#)
- [Allow access from specific virtual networks \(service endpoints\)](#)

- [Allow access via private endpoints](#)

In all cases, it's important to note that at least one IP firewall rule or virtual network rule for the namespace is specified. Otherwise, if no IP address or virtual network rule is specified, the namespace is accessible over the public internet (using the access key).

Azure Functions can be configured to consume events from or publish events to event hubs, which are set up with either service endpoints or private endpoints. Regional virtual network integration is needed for your function app to connect to an event hub using a service endpoint or a private endpoint.

When setting up Functions to work with a private endpoint enabled resource, you need to set the `WEBSITE_VNET_ROUTE_ALL` application setting to `1`. If you want to fully lock down your function app, you also need to [restrict your storage account](#).

To trigger (consume) events in a virtual network environment, the function app needs to be hosted in a Premium plan, a Dedicated (App Service) plan, or an App Service Environment (ASE).

Additionally, running in an Azure Functions Premium plan and consuming events from a virtual network restricted Event Hub requires virtual network trigger support, also referred to as [runtime scale monitoring](#). Runtime scale monitoring can be configured via the Azure portal, Azure CLI, or other deployment solutions. Runtime scale monitoring isn't available when the function is running in a Dedicated (App Service) plan or an ASE.

To use runtime scale monitoring with Event Hubs, you need to use version 4.1.0 or higher of the `Microsoft.Azure.WebJobs.Extensions.EventHubs` extension.

Next steps

Before continuing, consider reviewing these related articles:

- [Authorize access with Azure Active Directory](#)
- [Authorize access with a shared access signature in Azure Event Hubs](#)
- [Configure an identity-based resource](#)

Observability

Related resources

- [Monitoring serverless event processing](#) provides guidance on monitoring serverless event-driven architectures.
- [Serverless event processing](#) is a reference architecture detailing a typical architecture of this type, with code samples and discussion of important considerations.
- [De-batching and filtering in serverless event processing with Event Hubs](#) describes in more detail how these portions of the reference architecture work.

DevOps architecture design

3/10/2022 • 6 minutes to read • [Edit Online](#)

The term *DevOps* derives from *development* and *operations*. It refers to the integration of development, quality assurance, and IT operations into a unified culture and set of processes for delivering software. For an overview of DevOps, see [What is DevOps?](#).

DevOps includes these activities and operations:

- **Continuous integration (CI)** is the practice of merging all developer code into a central codebase frequently, and then performing automated build and test processes. The objectives are to quickly discover and correct code issues, to streamline deployment, and to ensure code quality. For more information, see [What is Continuous Integration?](#).
- **Continuous delivery (CD)** is the practice of automatically building, testing, and deploying code to production-like environments. The objective is to ensure that code is always ready to deploy. Adding continuous delivery to create a full CI/CD pipeline helps you detect code defects as soon as possible. It also ensures that properly tested updates can be released in a short time. For more information, see [What is Continuous Delivery?](#).
- **Continuous deployment** is an additional process that automatically takes any updates that have passed through the CI/CD pipeline and deploys them into production. Continuous deployment requires robust automatic testing and advanced process planning. It might not be appropriate for all teams.
- **Continuous monitoring** refers to the process and technology required to incorporate monitoring across each phase of DevOps and IT operations lifecycles. Monitoring helps to ensure the health, performance, and reliability of your application and infrastructure as the application moves from development to production. Continuous monitoring builds on the concepts of CI and CD.

Introduction to DevOps on Azure

If you need to know more about DevOps, or DevOps on Azure, the best place to learn is [Microsoft Learn](#). It's a free, online platform that provides interactive training for Microsoft products and more. There are videos, tutorials, and hands-on learning for specific products and services, plus learning paths based on job role, such as developer or data analyst. If you're not familiar with Learn you can take [a tour of Microsoft Learn](#) or [a quick video tour of Microsoft Learn](#).

After you're familiar with Azure, you can decide whether to follow learning paths specific to DevOps, such as:

- [Get started with Azure DevOps](#)
- [Deploy applications with Azure DevOps](#)
- [Build applications with Azure DevOps](#)

[Browse other Microsoft Learn training materials for DevOps](#)

Path to production

Plan your path to production by reviewing:

- [DevOps guides](#)
- [Azure services that are often used in implementing DevOps solutions](#)
- [Example DevOps architectures](#)

DevOps guides

ARTICLE OR SECTION	DESCRIPTION
DevOps checklist	A list of things to consider and do when you implement DevOps attitudes and methods in culture, development, testing, release, monitoring, and management.
Operational Excellence patterns	A list of design patterns for achieving Operational Excellence—one of the five pillars of the Microsoft Azure Well-Architected Framework —in a cloud environment. See Cloud Design Patterns for more patterns.
Advanced Azure Resource Manager template functionality	Some advanced examples of template use.
DevTest Labs guidance	A series of articles to help you use Azure Devtest Labs to provision development and test environments. The first article in the series is DevTest Labs in the enterprise .
Azure Monitor guidance	A series of articles to help you use Azure Monitor to monitor cloud environments. The first article in the series is Azure Monitor best practices - Planning your monitoring strategy and configuration .
Continuous integration and delivery for an Azure Synapse Analytics workspace	An outline of how to use an Azure DevOps release pipeline and GitHub Actions to automate the deployment of an Azure Synapse workspace to multiple environments.
DevOps for quantum computing	A discussion of the DevOps requirements for hybrid quantum applications.
Platform automation for Azure VMware Solution enterprise-scale scenario	An overview for deploying Azure VMware Solution, including guidance for operational automation.

Azure DevOps services

AZURE SERVICE	DOCUMENTATION	DESCRIPTION
Azure Artifacts	Azure Artifacts overview	Fully integrated package management for your CI/CD pipelines.
Azure DevOps	Azure DevOps documentation	Modern dev services for managing your development lifecycle end-to-end. It includes Azure Repos, Azure Pipelines, and Azure Artifacts.
Azure DevTest Labs	Azure DevTest Labs documentation	Reusable templates and artifacts for provisioning development and test environments.
Azure Lab Services	Azure Lab Services documentation	A tool for setting up and providing on-demand access to preconfigured virtual machines (VMs).
Azure Monitor	Azure Monitor documentation	Provides full observability into your applications, infrastructure, and network.

AZURE SERVICE	DOCUMENTATION	DESCRIPTION
Azure Pipelines	Azure Pipelines documentation	Helps you automate build and deployment by using cloud-hosted pipelines.
Azure Repos	Azure Repos documentation	Provides unlimited, cloud-hosted private Git repos for your project.
Azure Resource Manager	Azure Resource Manager documentation	Provides consistent deployment, organization, and control for resource management.
Azure Resource Manager templates (ARM templates)	ARM template documentation	Templates that you can use to define the infrastructure and configuration for your project.
Azure Test Plans	Azure Test Plans documentation	Provides planned and exploratory testing services for your apps.

Example DevOps architectures

The DevOps architectures are found in two sections:

SECTION	FIRST ARTICLE IN THE SECTION
Architectures	Automate multistage DevOps pipelines with Azure Pipelines
Solution ideas	CI/CD for Azure VMs

Here are some example architectures. For each one there's a list of the key Azure services used in the architecture.

ARCHITECTURE	DESCRIPTION	AZURE SERVICES USED
Automate multistage DevOps pipelines with Azure Pipelines	Use Azure DevOps REST APIs to build CI/CD pipelines.	Azure DevOps, Logic Apps, Azure Pipelines
Automated API deployments with APIOps	Apply GitOps and DevOps techniques to ensure quality APIs.	Azure Repos, API Management, Azure DevOps, Azure Pipelines, Azure Repos
Design a CI/CD pipeline using Azure DevOps	Build a CI/CD pipeline by using Azure DevOps and other services.	Azure Repos, Azure Test Plans, Azure Pipelines
Teacher-provisioned virtual labs in Azure	Teachers can easily set up virtual machines for students to work on class exercises.	Lab Services
Enterprise monitoring with Azure Monitor	Use Azure Monitor to achieve enterprise-level monitoring and centralized monitoring management.	Azure Monitor

Best practices

The [Microsoft Azure Well-Architected Framework](#) provides reference guidance and best practices that you can use to improve the quality of your architectures. The framework comprises five pillars: Reliability, Security, Cost

Optimization, Operational Excellence, and Performance Efficiency. Here's where to find documentation of the pillars:

- [Reliability](#)
- [Security](#)
- [Cost Optimization](#)
- [Operational Excellence](#)
- [Performance Efficiency](#)

The following articles are about best practices that are specific to DevOps and to some DevOps services.

DevOps

- [How Teams at Microsoft Embraced a DevOps Culture - Azure webinar series](#)
- [DevOps checklist](#)
- [Azure cloud migration best practices checklist](#)
- [Resiliency checklist for specific Azure services](#)
- [Continuous monitoring with Azure Monitor](#)
- [Monitoring best practices for reliability in Azure applications](#)
- [Overview of the Azure Security Benchmark \(v1\)](#)
- [Azure Identity Management and access control security best practices](#)
- [Security best practices](#)
- [Azure security best practices and patterns](#)
- [Azure operational security checklist](#)
- [Azure security baseline for API Management](#)
- [Secure development best practices on Azure](#)

Azure Artifacts

- [Azure Artifacts: best practices](#)

Azure Resource Manager

- [ARM template best practices](#)
- [Best practices for Bicep](#)

Stay current with DevOps

Stay current with Azure DevOps by monitoring these articles:

- [Azure DevOps Feature Timeline](#)
- [Azure DevOps documentation - what's new?](#)

Additional resources

Example solutions

- [Design a CI/CD pipeline using Azure DevOps](#)
- [Manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps](#)
- [Run containers in a hybrid environment](#)

AWS or GCP professionals

- [AWS to Azure services comparison - DevOps and application monitoring](#)
- [Google Cloud to Azure services comparison - DevOps and application monitoring](#)

DevOps checklist

3/10/2022 • 14 minutes to read • [Edit Online](#)

DevOps is the integration of development, quality assurance, and IT operations into a unified culture and set of processes for delivering software. Use this checklist as a starting point to assess your DevOps culture and process.

Culture

Ensure business alignment across organizations and teams. Conflicts over resources, purpose, goals, and priorities within an organization can be a risk to successful operations. Ensure that the business, development, and operations teams are aligned.

Ensure that the team understands the software lifecycle. Your team needs to understand the overall lifecycle of the application, and where the application is in that lifecycle. This helps all team members know what they should be doing now, and what they should be planning and preparing for in the future.

Reduce cycle time. Aim to minimize the time it takes to move from ideas to usable developed software. Limit the size and scope of individual releases to keep the test burden low. Automate the build, test, configuration, and deployment processes whenever possible. Clear any obstacles to communication among developers, and between developers and operations.

Review and improve processes. Your processes and procedures, both automated and manual, are never final. Set up regular reviews of current workflows, procedures, and documentation, with a goal of continual improvement.

Do proactive planning. Proactively plan for failure. Have processes in place to quickly identify problems when they occur, escalate to the correct team members to fix, and confirm resolution.

Learn from failures. Failures are inevitable, but it's important to learn from failures to avoid repeating them. If an operational failure occurs, triage the problem, document the cause and solution, and share any lessons learned. Whenever possible, update your build processes to automatically detect such failures in the future.

Optimize for speed and collect data. Every planned improvement is a hypothesis. Work in the smallest increments possible. Treat new ideas as experiments. Instrument the experiments so that you can collect production data to assess their effectiveness. Be prepared to fail fast if the hypothesis is wrong.

Allow time for learning. Both failures and successes provide opportunities for learning. Before you move on to new projects, allow time to gather the important lessons, and make sure those lessons are absorbed by your team. Also give the team time to build skills, experiment, and learn about new tools and techniques.

Document operations. Document all tools, processes, and automated tasks with the same level of quality as your product code. Document the current design and architecture of any systems you support, along with recovery processes and other maintenance procedures. Focus on the steps you actually do, not theoretically optimal processes. Regularly review and update the documentation. For code, make sure that meaningful comments are included, especially in public APIs. Use tools to generate code documentation automatically whenever possible.

Share knowledge. Documentation is only useful if people know that it exists and can find it. Ensure that documentation is organized and easily discoverable. Be creative: use brown bags (informal presentations), videos, or newsletters to share knowledge.

Development

Provide developers with production-like environments. If development and test environments don't match the production environment, it's hard to test and diagnose problems. Therefore, keep development and test environments as close to the production environment as possible. Make sure that test data is consistent with the data used in production, even if it's sample data and not real production data (for privacy or compliance reasons). Plan to generate and anonymize sample test data.

Ensure that all authorized team members can provision infrastructure and deploy the application. Setting up production-like resources and deploying the application shouldn't involve complicated manual tasks or detailed technical knowledge of the system. Anyone with the right permissions should be able to create or deploy production-like resources without going to the operations team.

This recommendation doesn't imply that anyone can push live updates to the production deployment. It's about reducing friction for the development and QA teams to create production-like environments.

Instrument the application for insight. To understand the health of your application, you need to know how it's performing and whether it's experiencing any errors or problems. Always include instrumentation as a design requirement, and build the instrumentation into the application from the start. Instrumentation must include event logging for root cause analysis, but also telemetry and metrics to monitor the health and usage of the application.

Track your technical debt. In many projects, release schedules are prioritized over code quality to one degree or another. Always track when this occurs. Document any shortcuts or other suboptimal implementations, and schedule time to revisit these issues.

Consider pushing updates directly to production. To reduce the overall release cycle time, consider pushing properly tested code commits directly to production. Use [feature toggles](#) to control which features are enabled. This allows you to move quickly from development to release, using the toggles to enable or disable features. Toggles are also useful when you perform tests like [canary releases](#), where a particular feature is deployed to a subset of the production environment.

Testing

Automate testing. Manually testing software is tedious and susceptible to error. Automate common testing tasks and integrate the tests into your build processes. Automated testing ensures consistent test coverage and reproducibility. Integrated UI tests should also be done by an automated tool. Azure offers development and test resources that can help you configure and run testing. For more information, see [Development and test](#).

Test for failures. If a system can't connect to a service, how does it respond? Can it recover when the service is available again? Make fault-injection testing a standard part of review on test and staging environments. When your test process and practices are mature, consider running these tests in production.

Test in production. The release process doesn't end with deployment to production. Have tests in place to ensure that deployed code works as expected. For deployments that are infrequently updated, schedule production testing as a regular part of maintenance.

Automate performance testing to identify performance problems early. The impact of a serious performance problem can be as severe as a bug in the code. Although automated functional tests can prevent application bugs, they might not detect performance problems. Define acceptable performance goals for metrics like latency, load times, and resource usage. Include automated performance tests in your release pipeline to make sure the application meets those goals.

Perform capacity testing. An application might work fine under test conditions and then have problems in production because of scale or resource limitations. Always define the maximum expected capacity and usage

limits. Test to make sure the application can handle those limits, but also test what happens when those limits are exceeded. Capacity testing should be done at regular intervals.

After the initial release, you should run performance and capacity tests whenever updates are made to production code. Use historical data to fine-tune tests and to determine what types of tests need to be done.

Perform automated security penetration testing. Ensuring your application is secure is as important as testing any other functionality. Make automated penetration testing a standard part of the build and deployment process. Schedule regular security tests and vulnerability scanning on deployed applications, monitoring for open ports, endpoints, and attacks. Automated testing doesn't remove the need for in-depth security reviews at regular intervals.

Perform automated business continuity testing. Develop tests for large-scale business continuity, including backup recovery and failover. Set up automated processes to perform these tests regularly.

Release

Automate deployments. Automate deploying the application to test, staging, and production environments. Automation enables faster and more reliable deployments, and ensures consistent deployments to any supported environment. It removes the risk of human error caused by manual deployments. It also makes it easy to schedule releases for convenient times, to minimize any effects of potential downtime. Have systems in place to detect any problems during rollout, and have an automated way to roll forward fixes or roll back changes.

Use continuous integration. Continuous integration (CI) is the practice of merging all developer code into a central codebase on a regular schedule, and then automatically performing standard build and test processes. CI ensures that an entire team can work on a codebase at the same time without conflicts. It also ensures that code defects are found as early as possible. Preferably, the CI process should run every time that code is committed or checked in. At the very least, it should run once per day.

Consider adopting a [trunk-based development model](#). In this model, developers commit to a single branch (the trunk). There's a requirement that commits never break the build. This model facilitates CI, because all feature work is done in the trunk, and any merge conflicts are resolved when the commit happens.

Consider using continuous delivery. Continuous delivery (CD) is the practice of ensuring that code is always ready to deploy, by automatically building, testing, and deploying code to production-like environments. Adding continuous delivery to create a full CI/CD pipeline will help you detect code defects as soon as possible. It also ensures that properly tested updates can be released in a short time.

Continuous *deployment* is an additional process that automatically takes any updates that have passed through the CI/CD pipeline and deploys them into production. Continuous deployment requires robust automatic testing and advanced process planning. It might not be appropriate for all teams.

Make small incremental changes. Large code changes have a greater potential to introduce bugs. Whenever possible, keep changes small. Doing so limits the potential effects of each change and makes it easier to understand and debug any problems.

Control exposure to changes. Make sure you're in control of when updates are visible to your end users. Consider using feature toggles to control when features are enabled for end users.

Implement release management strategies to reduce deployment risk. Deploying an application update to production always entails some risk. To minimize this risk, use strategies like [canary releases](#) or [blue/green deployments](#) to deploy updates to a subset of users. Confirm the update works as expected, and then roll the update out to the rest of the system.

Document all changes. Minor updates and configuration changes can be a source of confusion and versioning conflict. Always keep a clear record of any changes, no matter how small. Log everything that changes, including patches applied, policy changes, and configuration changes. (Don't include sensitive data in these logs. For example, log that a credential was updated, and who made the change, but don't record the updated credentials.) The record of the changes should be visible to the entire team.

Consider making infrastructure immutable. Immutable infrastructure is based on the principle that you shouldn't modify infrastructure after it's deployed to production. Otherwise, you can get into a state where ad hoc changes have been applied, making it hard to know exactly what changed. Immutable infrastructure works by replacing entire servers as part of any new deployment. This allows the code and the hosting environment to be tested and deployed as a block. After they're deployed, infrastructure components aren't modified until the next build and deploy cycle.

Monitoring

Make systems observable. The operations team should always have clear visibility into the health and status of a system or service. Set up external health endpoints to monitor status, and ensure that applications are coded to instrument the operations metrics. Use a common and consistent schema that helps you correlate events across systems. [Azure Diagnostics](#) and [Application Insights](#) are the standard method of tracking the health and status of Azure resources. [Azure Monitor](#) also provides centralized monitoring and management for cloud or hybrid solutions.

Aggregate and correlate logs and metrics. A properly instrumented telemetry system provides a large amount of raw performance data and event logs. Make sure that telemetry and log data is processed and correlated quickly, so that operations staff always has an up-to-date picture of system health. Organize and display data in ways that give a cohesive view of any problems, so that whenever possible it's clear when events are related to one another.

Consult your corporate retention policy for requirements on how data is processed and how long it should be stored.

Implement automated alerts and notifications. Set up monitoring tools like [Azure Monitor](#) to detect patterns or conditions that indicate potential or current problems. Send alerts to the team members who can address the problems. Tune the alerts to avoid false positives.

Monitor assets and resources for expirations. Some resources and assets, like certificates, expire. Be sure to track which assets expire, when they expire, and what services or features depend on them. Use automated processes to monitor these assets. Notify the operations team before an asset expires, and escalate if expiration threatens to disrupt the application.

Management

Automate operations tasks. Manually handling repetitive operations processes is error-prone. Automate these tasks whenever possible to ensure consistent execution and quality. Code that implements the automation should be versioned in source control. As with any other code, automation tools must be tested.

Take an infrastructure-as-code approach to provisioning. Minimize the amount of manual configuration needed to provision resources. Instead, use scripts and [Azure Resource Manager](#) templates. Keep the scripts and templates in source control, like any other code you maintain.

Consider using containers. Containers provide a standard package-based interface for deploying applications. When you use containers, you deploy the application by using self-contained packages that include any software, dependencies, and files that are needed to run the application. This greatly simplifies the deployment process.

Containers also create an abstraction layer between the application and the underlying operating system, which provides consistency across environments. This abstraction can also isolate a container from other processes or applications that run on a host.

Implement resiliency and self-healing. Resiliency is the ability of an application to recover from failures. Strategies for resiliency include retrying transient failures, and failing over to a secondary instance or even to another region. For more information, see [Designing reliable Azure applications](#). Instrument your applications so that problems are reported immediately and you can manage outages or other system failures.

Have an operations manual. An operations manual, or *runbook*, documents the procedures and management information needed for operations staff to maintain a system. Also document any operations scenarios and mitigation plans that might come into play during a failure or other disruption to your service. Create this documentation during the development process and keep it up-to-date afterwards. This is a living document, and should be reviewed, tested, and improved regularly.

Shared documentation is critical. Encourage team members to contribute and share knowledge. The entire team should have access to documents. Make it easy for anyone on the team to help keep documents updated.

Document on-call procedures. Make sure on-call duties, schedules, and procedures are documented and shared to all team members. Keep this information up-to-date at all times.

Document escalation procedures for third-party dependencies. If your application depends on external third-party services that you don't directly control, you need a plan to deal with outages. Create documentation for your planned mitigation processes. Include support contacts and escalation paths.

Use configuration management. Configuration changes should be planned, visible to operations, and recorded. This could take the form of a configuration management database, or a configuration-as-code approach. Configuration should be audited regularly to ensure that what's expected is actually in place.

Get an Azure support plan and understand the process. Azure offers a number of [support plans](#). Determine the right plan for your needs, and make sure the entire team knows how to use it. Team members should understand the details of the plan, how the support process works, and how to open a support ticket with Azure. If you're expecting a high-scale event, Azure support can assist you with increasing your service limits. For more information, see the [Azure support FAQs](#).

Follow least-privilege principles when granting access to resources. Carefully manage access to resources. Access should be denied by default, unless a user is explicitly given access to a resource. Only grant users access to what they need to complete their tasks. Track user permissions and perform regular security audits.

Use Azure role-based access control. Assigning user accounts and access to resources shouldn't be a manual process. Use [Azure role-based access control \(Azure RBAC\)](#) to grant access based on [Azure Active Directory](#) identities and groups.

Use a bug tracking system to track problems. Without a good way to track problems, it's easy to miss items, duplicate work, or introduce new problems. Don't rely on informal person-to-person communication to track the status of bugs. Use a bug tracking tool to record details about problems, assign resources to address them, and provide an audit trail of progress and status.

Manage all resources in a change management system. All aspects of your DevOps process should be included in a management and versioning system so that changes can be easily tracked and audited. This includes code, infrastructure, configuration, documentation, and scripts. Treat all these types of resources as code throughout the test, build, and review process.

Use checklists. Create operations checklists to ensure processes are followed. It's easy to miss something in a large manual, and following a checklist can force attention to details that might otherwise be overlooked. Maintain the checklists, and continually look for ways to automate tasks and streamline processes.

Next steps

- [What is DevOps?](#)
- [Azure DevOps documentation](#)
- [Microsoft Learn: Get started with Azure DevOps](#)
- [The DevOps journey at Microsoft](#)

Related resources

- [Design a CI/CD pipeline using Azure DevOps](#)
- [Automate multistage DevOps pipelines with Azure Pipelines](#)
- [CI/CD for Azure VMs](#)
- [CI/CD for containers](#)

Advanced Azure Resource Manager template functionality

3/10/2022 • 2 minutes to read • [Edit Online](#)

This section provides advanced examples for Azure Resource Manager templates.

Update a resource. You may need to update a resource during a deployment. You might encounter this scenario when you cannot specify all the properties for a resource until other, dependent resources are created.

Use an object parameter in a copy loop. There is a limit of 256 parameters per deployment. Once you get to larger and more complex deployments you may run out of parameters. One way to solve this problem is to use an object as a parameter instead of a value.

Property transformer and collector. A property transform and collector template can transform objects into the JSON schema expected by a nested template.

NOTE

These articles assume you have an advanced understanding of Azure Resource Manager templates.

Update a resource in an Azure Resource Manager template

3/10/2022 • 3 minutes to read • [Edit Online](#)

There are some scenarios in which you need to update a resource during a deployment. You might encounter this scenario when you cannot specify all the properties for a resource until other, dependent resources are created. For example, if you create a backend pool for a load balancer, you might update the network interfaces (NICs) on your virtual machines (VMs) to include them in the backend pool. And while Resource Manager supports updating resources during deployment, you must design your template correctly to avoid errors and to ensure the deployment is handled as an update.

First, you must reference the resource once in the template to create it and then reference the resource by the same name to update it later. However, if two resources have the same name in a template, Resource Manager throws an exception. To avoid this error, specify the updated resource in a second template that's either linked or included as a subtemplate using the `Microsoft.Resources/deployments` resource type.

Second, you must either specify the name of the existing property to change or a new name for a property to add in the nested template. You must also specify the original properties and their original values. If you fail to provide the original properties and values, Resource Manager assumes you want to create a new resource and deletes the original resource.

Example template

Let's look at an example template that demonstrates this. Our template deploys a virtual network named `firstVNet` that has one subnet named `firstSubnet`. It then deploys a virtual network interface (NIC) named `nic1` and associates it with our subnet. Then, a deployment resource named `updateVNet` includes a nested template that updates our `firstVNet` resource by adding a second subnet named `secondSubnet`.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "resources": [
    {
      "apiVersion": "2020-05-01",
      "name": "firstVNet",
      "location": "[resourceGroup().location]",
      "type": "Microsoft.Network/virtualNetworks",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "10.0.0.0/22"
          ]
        },
        "subnets": [
          {
            "name": "firstSubnet",
            "properties": {
              "addressPrefix": "10.0.0.0/24"
            }
          }
        ]
      }
    },
    {
      "name": "nic1",
      "type": "Microsoft.Network/networkInterfaces",
      "location": "[resourceGroup().location]",
      "dependsOn": [
        "[resourceId('Microsoft.Network/virtualNetworks', 'firstVNet')]"
      ],
      "properties": {
        "ipConfigurations": [
          {
            "name": "ipConfig1",
            "properties": {
              "subnet": {
                "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', 'firstVNet', 'firstSubnet')]"
              }
            }
          }
        ]
      }
    },
    {
      "name": "updateVNet",
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2020-05-01",
      "dependsOn": [
        "[resourceId('Microsoft.Network/virtualNetworks', 'firstVNet')]"
      ],
      "properties": {
        "template": {
          "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": {},
          "resources": [
            {
              "apiVersion": "2020-05-01",
              "name": "firstVNet",
              "type": "Microsoft.Network/virtualNetworks",
              "properties": {
                "subnets": [
                  {
                    "name": "firstSubnet",
                    "properties": {
                      "addressPrefix": "10.0.0.0/24"
                    }
                  },
                  {
                    "name": "secondSubnet",
                    "properties": {
                      "addressPrefix": "10.0.1.0/24"
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    }
  ]
}
```

```

    "apiVersion": "2020-05-01",
    "type": "Microsoft.Network/networkInterfaces",
    "name": "nic1",
    "location": "[resourceGroup().location]",
    "dependsOn": [
        "firstVNet"
    ],
    "properties": {
        "ipConfigurations": [
            {
                "name": "ipconfig1",
                "properties": {
                    "privateIPAllocationMethod": "Dynamic",
                    "subnet": {
                        "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', 'firstVNet',
'firstSubnet')]"
                    }
                }
            }
        ]
    }
},
{
    "apiVersion": "2020-06-01",
    "type": "Microsoft.Resources/deployments",
    "name": "updateVNet",
    "dependsOn": [
        "nic1"
    ],
    "properties": {
        "mode": "Incremental",
        "parameters": {},
        "template": {
            "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
            "contentVersion": "1.0.0.1",
            "parameters": {},
            "variables": {},
            "resources": [
                {
                    "apiVersion": "2020-05-01",
                    "name": "firstVNet",
                    "location": "[resourceGroup().location]",
                    "type": "Microsoft.Network/virtualNetworks",
                    "properties": {
                        "addressSpace": "[reference('firstVNet').addressSpace]",
                        "subnets": [
                            {
                                "name": "[reference('firstVNet').subnets[0].name]",
                                "properties": {
                                    "addressPrefix": "
[reference('firstVNet').subnets[0].properties.addressPrefix]"
                                }
                            },
                            {
                                "name": "secondSubnet",
                                "properties": {
                                    "addressPrefix": "10.0.1.0/24"
                                }
                            }
                        ]
                    }
                },
                {
                    "outputs": {}
                }
            ]
        }
    }
],

```

```
        "outputs": {}  
    }
```

Let's take a look at the resource object for our `firstVNet` resource first. Notice that we specify again the settings for our `firstVNet` in a nested template—this is because Resource Manager doesn't allow the same deployment name within the same template and nested templates are considered to be a different template. By again specifying our values for our `firstSubnet` resource, we are telling Resource Manager to update the existing resource instead of deleting it and redeploying it. Finally, our new settings for `secondSubnet` are picked up during this update.

Try the template

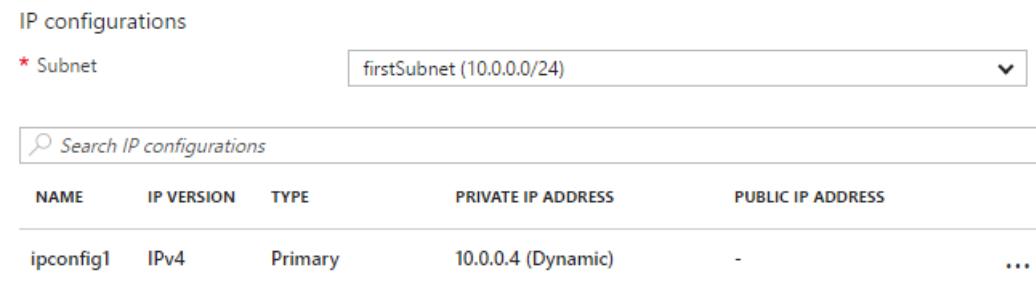
An example template is available on [GitHub](#). To deploy the template, run the following [Azure CLI](#) commands:

```
az group create --location <location> --name <resource-group-name>  
az deployment group create -g <resource-group-name> \  
  --template-uri https://raw.githubusercontent.com/mspnp/template-examples/master/example1-  
  update/deploy.json
```

Once deployment has finished, open the resource group you specified in the portal. You see a virtual network named `firstVNet` and a NIC named `nic1`. Click `firstVNet`, then click `subnets`. You see the `firstSubnet` that was originally created, and you see the `secondSubnet` that was added in the `updateVNet` resource.

NAME	ADDRESS RANGE	AVAILABLE ADDRESSES	SECURITY GROUP	
firstSubnet	10.0.0.0/24	250	-	...
secondSubnet	10.0.1.0/24	251	-	...

Then, go back to the resource group and click `nic1`, then click `IP configurations`. In the `IP configurations` section, the `subnet` is set to `firstSubnet (10.0.0.0/24)`.



IP configurations

* Subnet

Search IP configurations

NAME	IP VERSION	TYPE	PRIVATE IP ADDRESS	PUBLIC IP ADDRESS
ipconfig1	IPv4	Primary	10.0.0.4 (Dynamic)	-

The original `firstVNet` has been updated instead of re-created. If `firstVNet` had been re-created, `nic1` would not be associated with `firstVNet`.

Next steps

- Learn how to [Use an object as a parameter in an Azure Resource Manager template](#).

Use objects as parameters in a copy loop in an Azure Resource Manager template

3/10/2022 • 2 minutes to read • [Edit Online](#)

When [using objects as a parameter in Azure Resource Manager templates](#) you may want to include them in a copy loop, so here is an example that uses them in that way:

This approach becomes very useful when combined with the [serial copy loop](#), particularly for deploying child resources.

To demonstrate this, let's look at a template that deploys a [network security group \(NSG\)](#) with two security rules.

First, let's take a look at our parameters. When we look at our template we'll see that we've defined one parameter named `networkSecurityGroupsSettings` that includes an array named `securityRules`. This array contains two JSON objects that specify a number of settings for a security rule.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters":{  
        "networkSecurityGroupsSettings": {  
            "value": {  
                "securityRules": [  
                    {  
                        "name": "RDPAAllow",  
                        "description": "allow RDP connections",  
                        "direction": "Inbound",  
                        "priority": 100,  
                        "sourceAddressPrefix": "*",  
                        "destinationAddressPrefix": "10.0.0.0/24",  
                        "sourcePortRange": "*",  
                        "destinationPortRange": "3389",  
                        "access": "Allow",  
                        "protocol": "Tcp"  
                    },  
                    {  
                        "name": "HTTPAllow",  
                        "description": "allow HTTP connections",  
                        "direction": "Inbound",  
                        "priority": 200,  
                        "sourceAddressPrefix": "*",  
                        "destinationAddressPrefix": "10.0.1.0/24",  
                        "sourcePortRange": "*",  
                        "destinationPortRange": "80",  
                        "access": "Allow",  
                        "protocol": "Tcp"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Now let's take a look at our template. We have a resource named `NSG1` that deploys the NSG, it also leverages [ARM's built-in property iteration feature](#); by adding copy loop to the properties section of a resource in your template, you can dynamically set the number of items for a property during deployment. You also avoid having to repeat template syntax.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "VNetSettings": {
            "type": "object"
        },
        "networkSecurityGroupsSettings": {
            "type": "object"
        }
    },
    "resources": [
        {
            "apiVersion": "2020-05-01",
            "type": "Microsoft.Network/virtualNetworks",
            "name": "[parameters('VNetSettings').name]",
            "location": "[resourceGroup().location]",
            "properties": {
                "addressSpace": {
                    "addressPrefixes": [
                        "[parameters('VNetSettings').addressPrefixes[0].addressPrefix]"
                    ]
                },
                "subnets": [
                    {
                        "name": "[parameters('VNetSettings').subnets[0].name]",
                        "properties": {
                            "addressPrefix": "[parameters('VNetSettings').subnets[0].addressPrefix]"
                        }
                    },
                    {
                        "name": "[parameters('VNetSettings').subnets[1].name]",
                        "properties": {
                            "addressPrefix": "[parameters('VNetSettings').subnets[1].addressPrefix]"
                        }
                    }
                ]
            }
        },
        {
            "apiVersion": "2020-05-01",
            "type": "Microsoft.Network/networkSecurityGroups",
            "name": "NSG1",
            "location": "[resourceGroup().location]",
            "properties": {
                "copy": [
                    {
                        "name": "securityRules",
                        "count": "[length(parameters('networkSecurityGroupsSettings').securityRules)]",
                        "input": {
                            "description": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].description]",
                            "priority": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].priority]",
                            "protocol": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].protocol]",
                            "sourcePortRange": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].sourcePortRange]",
                            "destinationPortRange": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].destinationPortRange]",
                            "sourceAddressPrefix": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].sourceAddressPrefix]",
                            "destinationAddressPrefix": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].destinationAddressPrefix]",
                            "access": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].access]",
                            "direction": "[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].direction]"
                        }
                    }
                ]
            }
        }
    ]
}
```

```
        }
      ]
    }
  ]
}
```

Let's take a closer look at how we specify our property values in the `securityRules` child resource. All of our properties are referenced using the `parameters()` function, and then we use the dot operator to reference our `securityRules` array, indexed by the current value of the iteration. Finally, we use another dot operator to reference the name of the object.

Try the template

An example template is available on [GitHub](#). To deploy the template, clone the repo and run the following [Azure CLI](#) commands:

```
git clone https://github.com/mspnp/template-examples.git
cd template-examples/example3-object-param
az group create --location <location> --name <resource-group-name>
az deployment group create -g <resource-group-name> \
  --template-uri https://raw.githubusercontent.com/mspnp/template-examples/master/example3-object-
param/deploy.json \
  --parameters deploy.parameters.json
```

Next steps

- Learn how to create a template that iterates through an object array and transforms it into a JSON schema.
See [Implement a property transformer and collector in an Azure Resource Manager template](#)

Implement a property transformer and collector in an Azure Resource Manager template

3/10/2022 • 6 minutes to read • [Edit Online](#)

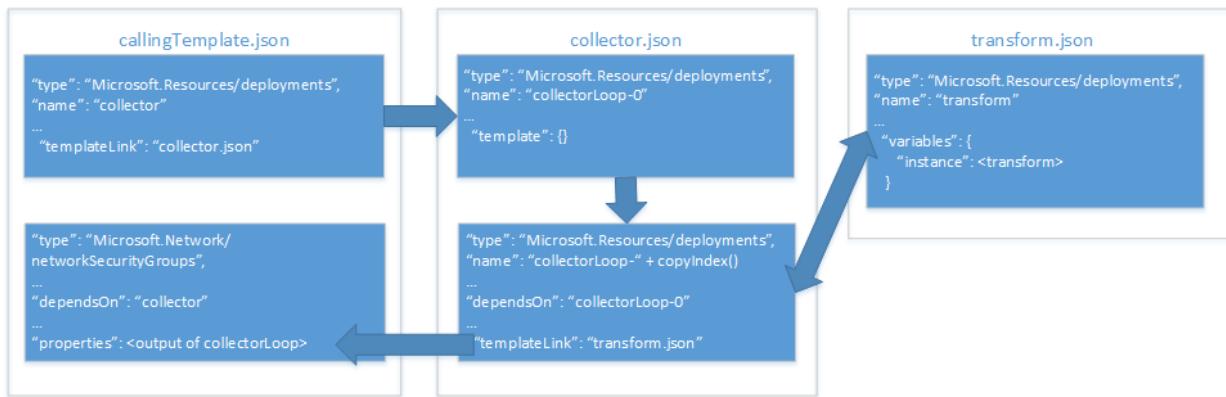
In [use an object as a parameter in an Azure Resource Manager template](#), you learned how to store resource property values in an object and apply them to a resource during deployment. While this is a very useful way to manage your parameters, it still requires you to map the object's properties to resource properties each time you use it in your template.

To work around this, you can implement a property transform and collector template that iterates your object array and transforms it into the JSON schema expected by the resource.

IMPORTANT

This approach requires that you have a deep understanding of Resource Manager templates and functions.

Let's take a look at how we can implement a property collector and transformer with an example that deploys a [network security group](#). The diagram below shows the relationship between our templates and our resources within those templates:



Our **calling template** includes two resources:

- A template link that invokes our **collector template**.
- The network security group resource to deploy.

Our **collector template** includes two resources:

- An **anchor** resource.
- A template link that invokes the transform template in a copy loop.

Our **transform template** includes a single resource: an empty template with a variable that transforms our **source** JSON to the JSON schema expected by our network security group resource in the **main template**.

Parameter object

We'll be using our **securityRules** parameter object from [objects as parameters](#). Our **transform template** will transform each object in the **securityRules** array into the JSON schema expected by the network security group resource in our **calling template**.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "networkSecurityGroupsSettings": {
            "value": {
                "securityRules": [
                    {
                        "name": "RDPAllow",
                        "description": "allow RDP connections",
                        "direction": "Inbound",
                        "priority": 100,
                        "sourceAddressPrefix": "*",
                        "destinationAddressPrefix": "10.0.0.0/24",
                        "sourcePortRange": "*",
                        "destinationPortRange": "3389",
                        "access": "Allow",
                        "protocol": "Tcp"
                    },
                    {
                        "name": "HTTPAllow",
                        "description": "allow HTTP connections",
                        "direction": "Inbound",
                        "priority": 200,
                        "sourceAddressPrefix": "*",
                        "destinationAddressPrefix": "10.0.1.0/24",
                        "sourcePortRange": "*",
                        "destinationPortRange": "80",
                        "access": "Allow",
                        "protocol": "Tcp"
                    }
                ]
            }
        }
    }
}
```

Let's look at our **transform template** first.

Transform template

Our **transform template** includes two parameters that are passed from the **collector template**:

- **source** is an object that receives one of the property value objects from the property array. In our example, each object from the `"securityRules"` array will be passed in one at a time.
- **state** is an array that receives the concatenated results of all the previous transforms. This is the collection of transformed JSON.

Our parameters look like this:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "source": {
            "type": "object"
        },
        "state": {
            "type": "array",
            "defaultValue": []
        }
    }
},
```

Our template also defines a variable named `instance`. It performs the actual transform of our `source` object into the required JSON schema:

```
"variables": {
  "instance": [
    {
      "name": "[parameters('source').name]",
      "properties": {
        "description": "[parameters('source').description]",
        "protocol": "[parameters('source').protocol]",
        "sourcePortRange": "[parameters('source').sourcePortRange]",
        "destinationPortRange": "[parameters('source').destinationPortRange]",
        "sourceAddressPrefix": "[parameters('source').sourceAddressPrefix]",
        "destinationAddressPrefix": "[parameters('source').destinationAddressPrefix]",
        "access": "[parameters('source').access]",
        "priority": "[parameters('source').priority]",
        "direction": "[parameters('source').direction]"
      }
    }
  ]
}
```

Finally, the `output` of our template concatenates the collected transforms of our `state` parameter with the current transform performed by our `instance` variable:

```
"resources": [],
"outputs": {
  "collection": {
    "type": "array",
    "value": "[concat(parameters('state'), variables('instance'))]"
  }
}
```

Next, let's take a look at our **collector template** to see how it passes in our parameter values.

Collector template

Our **collector template** includes three parameters:

- `source` is our complete parameter object array. It's passed in by the **calling template**. This has the same name as the `source` parameter in our **transform template** but there is one key difference that you may have already noticed: this is the complete array, but we only pass one element of this array to the **transform template** at a time.
- `transformTemplateUri` is the URI of our **transform template**. We're defining it as a parameter here for template reusability.
- `state` is an initially empty array that we pass to our **transform template**. It stores the collection of transformed parameter objects when the copy loop is complete.

Our parameters look like this:

```
"parameters": {  
    "source": {  
        "type": "array"  
    },  
    "transformTemplateUri": {  
        "type": "string"  
    },  
    "state": {  
        "type": "array",  
        "defaultValue": []  
    }  
}
```

Next, we define a variable named `count`. Its value is the length of the `source` parameter object array:

```
"variables": {  
    "count": "[length(parameters('source'))]"  
}
```

As you might suspect, we use it for the number of iterations in our copy loop.

Now let's take a look at our resources. We define two resources:

- `loop-0` is the zero-based resource for our copy loop.
- `loop-` is concatenated with the result of the `copyIndex(1)` function to generate a unique iteration-based name for our resource, starting with `1`.

Our resources look like this:

```

"resources": [
    {
        "type": "Microsoft.Resources/deployments",
        "apiVersion": "2015-01-01",
        "name": "loop-0",
        "properties": {
            "mode": "Incremental",
            "parameters": { },
            "template": {
                "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
                "contentVersion": "1.0.0.0",
                "parameters": { },
                "variables": { },
                "resources": [ ],
                "outputs": {
                    "collection": {
                        "type": "array",
                        "value": "[parameters('state')]"
                    }
                }
            }
        },
        {
            "type": "Microsoft.Resources/deployments",
            "apiVersion": "2015-01-01",
            "name": "[concat('loop-', copyIndex(1))]",
            "copy": {
                "name": "iterator",
                "count": "[variables('count')]",
                "mode": "serial"
            },
            "dependsOn": [
                "loop-0"
            ],
            "properties": {
                "mode": "Incremental",
                "templateLink": { "uri": "[parameters('transformTemplateUri')]" },
                "parameters": {
                    "source": { "value": "[parameters('source')[copyIndex()]]" },
                    "state": { "value": "[reference(concat('loop-', copyIndex())).outputs.collection.value]" }
                }
            }
        }
    }
]

```

Let's take a closer look at the parameters we're passing to our **transform template** in the nested template. Recall from earlier that our `source` parameter passes the current object in the `source` parameter object array. The `state` parameter is where the collection happens, because it takes the output of the previous iteration of our copy loop—notice that the `reference()` function uses the `copyIndex()` function with no parameter to reference the `name` of our previous linked template object—and passes it to the current iteration.

Finally, the `output` of our template returns the `output` of the last iteration of our **transform template**:

```

"outputs": {
    "result": {
        "type": "array",
        "value": "[reference(concat('loop-', variables('count'))).outputs.collection.value]"
    }
}

```

It may seem counterintuitive to return the `output` of the last iteration of our **transform template** to our

calling template because it appeared we were storing it in our `source` parameter. However, remember that it's the last iteration of our **transform template** that holds the complete array of transformed property objects, and that's what we want to return.

Finally, let's take a look at how to call the **collector template** from our **calling template**.

Calling template

Our **calling template** defines a single parameter named `networkSecurityGroupsSettings`:

```
...
"parameters": {
    "networkSecurityGroupsSettings": {
        "type": "object"
    }
}
```

Next, our template defines a single variable named `collectorTemplateUri`:

```
"variables": {
    "collectorTemplateUri": "[uri(deployment().properties.templateLink.uri, 'collector.template.json')]"
}
```

As you would expect, this is the URI for the **collector template** that will be used by our linked template resource:

```
{
    "apiVersion": "2020-06-01",
    "name": "collector",
    "type": "Microsoft.Resources/deployments",
    "properties": {
        "mode": "Incremental",
        "templateLink": {
            "uri": "[variables('collectorTemplateUri')]",
            "contentVersion": "1.0.0.0"
        },
        "parameters": {
            "source": {
                "value": "[parameters('networkSecurityGroupsSettings').securityRules]"
            },
            "transformTemplateUri": {
                "value": "[uri(deployment().properties.templateLink.uri, 'transform.json')]"
            }
        }
    }
}
```

We pass two parameters to the **collector template**:

- `source` is our property object array. In our example, it's our `networkSecurityGroupsSettings` parameter.
- `transformTemplateUri` is the variable we just defined with the URI of our **collector template**.

Finally, our `Microsoft.Network/networkSecurityGroups` resource directly assigns the `output` of the `collector` linked template resource to its `securityRules` property:

```
"resources": [
    {
        "apiVersion": "2020-05-01",
        "type": "Microsoft.Network/networkSecurityGroups",
        "name": "networkSecurityGroup1",
        "location": "[resourceGroup().location]",
        "properties": {
            "securityRules": "[reference('collector').outputs.result.value]"
        }
    }
],
"outputs": {
    "instance": {
        "type": "array",
        "value": "[reference('collector').outputs.result.value]"
    }
}
```

Try the template

An example template is available on [GitHub](#). To deploy the template, clone the repo and run the following [Azure CLI](#) commands:

```
git clone https://github.com/mspnp/template-examples.git
cd template-examples/example4-collector
az group create --location <location> --name <resource-group-name>
az deployment group create -g <resource-group-name> \
    --template-uri https://raw.githubusercontent.com/mspnp/template-examples/master/example4-
collector/deploy.json \
    --parameters deploy.parameters.json
```


Hybrid architecture design

3/10/2022 • 2 minutes to read • [Edit Online](#)

Many organizations need a hybrid approach to analytics, automation, and services because their data is hosted both on-premises and in the cloud. Organizations often [extend on-premises data solutions to the cloud](#). To connect environments, organizations start by [choosing a hybrid network architecture](#).

Learn about hybrid solutions

If you're new to Azure, the best place to start is with Microsoft Learn. Microsoft Learn is a free, online training platform that provides interactive learning for Microsoft products and more. The [Introduction to Azure hybrid cloud services](#) module helps you build foundational knowledge and takes you through core concepts.

[Browse other hybrid solutions in Microsoft Learn](#)

Path to production

Explore some options for [connecting an on-premises network to Azure](#):

- [Extend an on-premises network using VPN](#)
- [Extend an on-premises network using ExpressRoute](#)
- [Connect an on-premises network to Azure using ExpressRoute](#)

Best practices

When you adopt a hybrid model, you can choose from multiple solutions to confidently deliver hybrid workloads. See these documents for information on running Azure data services anywhere, modernizing applications anywhere, and managing your workloads anywhere:

- [Azure Automation in a hybrid environment](#)
- [Azure Arc hybrid management and deployment for Kubernetes clusters](#)
- [Run containers](#)
- [Use Azure file shares](#)
- [Back up files](#)
- [Manage workloads](#)
- [Monitor performance](#)
- [Disaster recovery for Azure Stack Hub VMs](#)

Additional resources

The typical hybrid solution journey ranges from learning how to get started with a hybrid architecture to how to use Azure services in hybrid environments. However, you might also just be looking for additional reference and supporting material to help along the way for your specific situation. See these resources for general information on hybrid architectures:

- [Browse hybrid and multicloud architectures](#)
- [Troubleshoot a hybrid VPN connection](#)

Example solutions

Here are some example implementations to consider:

- [Cross-cloud scaling](#)
- [Cross-platform chat](#)
- [Hybrid connections](#)
- [Unlock legacy data with Azure Stack](#)

Configure hybrid cloud connectivity using Azure and Azure Stack Hub

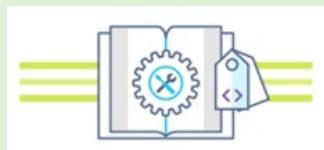
3/10/2022 • 9 minutes to read • [Edit Online](#)

You can access resources with security in global Azure and Azure Stack Hub using the hybrid connectivity pattern.

In this solution, you'll build a sample environment to:

- Keep data on-premises to meet privacy or regulatory requirements but keep access to global Azure resources.
- Maintain a legacy system while using cloud-scaled app deployment and resources in global Azure.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that allows you to build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Prerequisites

A few components are required to build a hybrid connectivity deployment. Some of these components take time to prepare, so plan accordingly.

Azure

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- Create a [web app](#) in Azure. Make note of the web app URL because you'll need it in the solution.

Azure Stack Hub

An Azure OEM/hardware partner can deploy a production Azure Stack Hub, and all users can deploy an Azure Stack Development Kit (ASDK).

- Use your production Azure Stack Hub or deploy the ASDK.

NOTE

Deploying the ASDK can take up to 7 hours, so plan accordingly.

- Deploy [App Service](#) PaaS services to Azure Stack Hub.
- [Create plans and offers](#) in the Azure Stack Hub environment.
- [Create tenant subscription](#) within the Azure Stack Hub environment.

Azure Stack Hub components

An Azure Stack Hub operator must deploy the App Service, create plans and offers, create a tenant subscription, and add the Windows Server 2016 image. If you already have these components, make sure they meet the requirements before you start this solution.

This solution example assumes that you have some basic knowledge of Azure and Azure Stack Hub. To learn more before starting the solution, read the following articles:

- [Introduction to Azure](#)
- [Azure Stack Hub Key Concepts](#)

Before you begin

Verify that you meet the following criteria before you start configuring hybrid cloud connectivity:

- You need an externally facing public IPv4 address for your VPN device. This IP address can't be located behind a NAT (Network Address Translation).
- All resources are deployed in the same region/location.

Solution example values

The examples in this solution use the following values. You can use these values to create a test environment or refer to them for a better understanding of the examples. For more information about VPN gateway settings, see [About VPN Gateway Settings](#).

Connection specifications:

- **VPN type:** route-based
- **Connection type:** site-to-site (IPsec)
- **Gateway type:** VPN
- **Azure connection name:** Azure-Gateway-AzureStack-S2SGateway (the portal will autofill this value)
- **Azure Stack Hub connection name:** AzureStack-Gateway-Azure-S2SGateway (the portal will autofill this value)
- **Shared key:** any compatible with VPN hardware, with matching values on both sides of connection
- **Subscription:** any preferred subscription
- **Resource group:** Test-Infra

Network and subnet IP addresses:

AZURE/AZURE STACK HUB CONNECTION	NAME	SUBNET	IP ADDRESS
Azure vNet	ApplicationvNet 10.100.102.9/23	ApplicationSubnet 10.100.102.0/24	
		GatewaySubnet 10.100.103.0/24	
Azure Stack Hub vNet	ApplicationvNet 10.100.100.0/23	ApplicationSubnet 10.100.100.0/24	
		GatewaySubnet 10.100.101.0/24	
Azure Virtual Network Gateway	Azure-Gateway		

AZURE/AZURE STACK HUB CONNECTION	NAME	SUBNET	IP ADDRESS
Azure Stack Hub Virtual Network Gateway	AzureStack-Gateway		
Azure Public IP	Azure-GatewayPublicIP		Determined at creation
Azure Stack Hub Public IP	AzureStack-GatewayPublicIP		Determined at creation
Azure Local Network Gateway	AzureStack-S2SGateway 10.100.100.0/23		Azure Stack Hub Public IP Value
Azure Stack Hub Local Network Gateway	Azure-S2SGateway 10.100.102.0/23		Azure Public IP Value

Create a virtual network in global Azure and Azure Stack Hub

Use the following steps to create a virtual network by using the portal. You can use these [example values](#) if you're using this article as only a solution. If you're using this article to configure a production environment, replace the example settings with your own values.

IMPORTANT

You must ensure that there isn't an overlap of IP addresses in Azure or Azure Stack Hub vNet address spaces.

To create a vNet in Azure:

1. Use your browser to connect to the [Azure portal](#) and sign in with your Azure account.
2. Select **Create a resource**. In the **Search the marketplace** field, enter 'virtual network'. Select **Virtual network** from the results.
3. From the **Select a deployment model** list, select **Resource Manager**, and then select **Create**.
4. On **Create virtual network**, configure the VNet settings. The required fields names are prefixed with a red asterisk. When you enter a valid value, the asterisk changes to a green check mark.

To create a vNet in Azure Stack Hub:

1. Repeat the steps above (1-4) using the Azure Stack Hub [tenant portal](#).

Add a gateway subnet

Before connecting your virtual network to a gateway, you need to create the gateway subnet for the virtual network that you want to connect to. The gateway services use the IP addresses you specify in the gateway subnet.

In the [Azure portal](#), navigate to the Resource Manager virtual network where you want to create a virtual network gateway.

1. Select the vNet to open the **Virtual network** page.
2. In **SETTINGS**, select **Subnets**.
3. On the **Subnets** page, select **+Gateway subnet** to open the **Add subnet** page.

The screenshot shows the Azure portal interface for creating a subnet. At the top, there are two buttons: '+ Subnet' and '+ Gateway subnet'. The '+ Gateway subnet' button is highlighted with a red box. Below these buttons is a search bar labeled 'Search subnets'. The main area displays a table with three columns: 'NAME', 'ADDRESS RANGE', and 'AVAILABLE ADDRESSES'. The table has a header row with sorting arrows for each column.

4. The **Name** for the subnet is automatically filled in with the value 'GatewaySubnet'. This value is required for Azure to recognize the subnet as the gateway subnet.
5. Change the **Address range** values that are provided to match your configuration requirements and then select **OK**.

Create a Virtual Network Gateway in Azure and Azure Stack

Use the following steps to create a virtual network gateway in Azure.

1. On the left side of the portal page, select + and enter 'virtual network gateway' in the search field.
2. In **Results**, select **Virtual network gateway**.
3. In **Virtual network gateway**, select **Create** to open the **Create virtual network gateway** page.
4. On **Create virtual network gateway**, specify the values for your network gateway using our **Tutorial example values**. Include the following additional values:
 - **SKU:** basic
 - **Virtual Network:** Select the virtual network you created earlier. The gateway subnet you created is automatically selected.
 - **First IP Configuration:** The public IP of your gateway.
 - Select **Create gateway IP configuration**, which takes you to the **Choose public IP address** page.
 - Select **+Create new** to open the **Create public IP address** page.
 - Enter a **Name** for your public IP address. Leave the SKU as **Basic**, and then select **OK** to save your changes.

NOTE

Currently, VPN Gateway only supports Dynamic Public IP address allocation. However, this doesn't mean that the IP address changes after it's assigned to your VPN gateway. The only time the public IP address changes is when the gateway is deleted and re-created. Resizing, resetting, or other internal maintenance/upgrades to your VPN gateway don't change the IP address.

5. Verify your gateway settings.
6. Select **Create** to create the VPN gateway. The gateway settings are validated and the "Deploying Virtual network gateway" tile is shown on your dashboard.

NOTE

Creating a gateway can take up to 45 minutes. You may need to refresh your portal page to see the completed status.

After the gateway is created, you can see the IP address assigned to it by looking at the virtual network in the portal. The gateway appears as a connected device. To see more information about the gateway, select the device.

7. Repeat the previous steps (1-5) on your Azure Stack Hub deployment.

Create the local network gateway in Azure and Azure Stack Hub

The local network gateway typically refers to your on-premises location. You give the site a name that Azure or Azure Stack Hub can refer to, and then specify:

- The IP address of the on-premises VPN device that you're creating a connection for.
- The IP address prefixes that will be routed through the VPN gateway to the VPN device. The address prefixes you specify are the prefixes located on your on-premises network.

NOTE

If your on-premises network changes or you need to change the public IP address for the VPN device, you can update these values later.

1. In the portal, select **+Create a resource**.
2. In the search box, enter **Local network gateway**, then select **Enter** to search. A list of results will display.
3. Select **Local network gateway**, then select **Create** to open the **Create local network gateway** page.
4. On **Create local network gateway**, specify the values for your local network gateway using our **Tutorial example values**. Include the following additional values:
 - **IP address**: The public IP address of the VPN device that you want Azure or Azure Stack Hub to connect to. Specify a valid public IP address that isn't behind a NAT so Azure can reach the address. If you don't have the IP address right now, you can use a value from the example as a placeholder. You'll have to go back and replace the placeholder with the public IP address of your VPN device. Azure can't connect to the device until you provide a valid address.
 - **Address Space**: the address range for the network that this local network represents. You can add multiple address space ranges. Make sure that the ranges you specify don't overlap with ranges of other networks that you want to connect to. Azure will route the address range that you specify to the on-premises VPN device IP address. Use your own values if you want to connect to your on-premises site, not an example value.
 - **Configure BGP settings**: Use only when configuring BGP. Otherwise, don't select this option.
 - **Subscription**: Verify that the correct subscription is showing.
 - **Resource Group**: Select the resource group that you want to use. You can either create a new resource group or select one that you've already created.
 - **Location**: Select the location that this object will be created in. You may want to select the same location that your VNet resides in, but you're not required to do so.
5. When you finish specifying the required values, select **Create** to create the local network gateway.
6. Repeat these steps (1-5) on your Azure Stack Hub deployment.

Configure your connection

Site-to-site connections to an on-premises network require a VPN device. The VPN device you configure is referred to as a connection. To configure your connection, you need:

- A shared key. This key is the same shared key that you specify when creating your site-to-site VPN connection. In our examples, we use a basic shared key. We recommend that you generate a more complex key to use.

- The public IP address of your virtual network gateway. You can view the public IP address by using the Azure portal, PowerShell, or CLI. To find the public IP address of your VPN gateway using the Azure portal, go to virtual network gateways, then select the name of your gateway.

Use the following steps to create a site-to-site VPN connection between your virtual network gateway and your on-premises VPN device.

1. In the Azure portal, select **+Create a resource**.
2. Search for **connections**.
3. In **Results**, select **Connections**.
4. On **Connection**, select **Create**.
5. On **Create Connection**, configure the following settings:
 - **Connection type:** Select site-to-site (IPSec).
 - **Resource Group:** Select your test resource group.
 - **Virtual Network Gateway:** Select the virtual network gateway you created.
 - **Local Network Gateway:** Select the local network gateway you created.
 - **Connection Name:** This name is autopopulated using the values from the two gateways.
 - **Shared Key:** This value must match the value that you're using for your local on-premises VPN device. The tutorial example uses 'abc123', but you should use something more complex. The important thing is that this value *must* be the same value that you specify when configuring your VPN device.
 - The values for **Subscription**, **Resource Group**, and **Location** are fixed.
6. Select **OK** to create your connection.

You can see the connection in the **Connections** page of the virtual network gateway. The status will go from *Unknown* to *Connecting*, and then to *Succeeded*.

Next steps

- To learn more about Azure Cloud Patterns, see [Cloud Design Patterns](#).

Configure hybrid cloud identity for Azure and Azure Stack Hub apps

3/10/2022 • 2 minutes to read • [Edit Online](#)

Learn how to configure a hybrid cloud identity for your Azure and Azure Stack Hub apps.

You have two options for granting access to your apps in both global Azure and Azure Stack Hub.

- When Azure Stack Hub has a continuous connection to the internet, you can use Azure Active Directory (Azure AD).
- When Azure Stack Hub is disconnected from the internet, you can use Azure Directory Federated Services (AD FS).

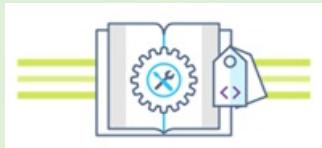
You use service principals to grant access to your Azure Stack Hub apps for deployment or configuration using the Azure Resource Manager in Azure Stack Hub.

In this solution, you'll build a sample environment to:

- Establish a hybrid identity in global Azure and Azure Stack Hub
- Retrieve a token to access the Azure Stack Hub API.

You must have Azure Stack Hub operator permissions for the steps in this solution.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that lets you build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Create a service principal for Azure AD in the portal

If you deployed Azure Stack Hub using Azure AD as the identity store, you can create service principals just like you do for Azure. [Use an app identity to access resources](#) shows you how to perform the steps through the portal. Be sure you have the [required Azure AD permissions](#) before beginning.

Create a service principal for AD FS using PowerShell

If you deployed Azure Stack Hub with AD FS, you can use PowerShell to create a service principal, assign a role for access, and sign in from PowerShell using that identity. [Use an app identity to access resources](#) shows you how to perform the required steps using PowerShell.

Using the Azure Stack Hub API

The [Azure Stack Hub API](#) solution walks you through the process of retrieving a token to access the Azure Stack

Hub API.

Connect to Azure Stack Hub using PowerShell

The quickstart [to get up and running with PowerShell in Azure Stack Hub](#) walks you through the steps needed to install Azure PowerShell and connect to your Azure Stack Hub installation.

Prerequisites

You need an Azure Stack Hub installation connected to Azure AD with a subscription you can access. If you don't have an Azure Stack Hub installation, you can use these instructions to set up an [Azure Stack Development Kit \(ASDK\)](#).

Connect to Azure Stack Hub using code

To connect to Azure Stack Hub using code, use the Azure Resource Manager endpoints API to get the authentication and graph endpoints for your Azure Stack Hub installation. Then authenticate using REST requests. You can find a sample client application on [GitHub](#).

NOTE

Unless the Azure SDK for your language of choice supports Azure API Profiles, the SDK may not work with Azure Stack Hub. To learn more about Azure API Profiles, see the [manage API version profiles](#) article.

Next steps

- To learn more about how identity is handled in Azure Stack Hub, see [Identity architecture for Azure Stack Hub](#).
- To learn more about Azure Cloud Patterns, see [Cloud Design Patterns](#).

Deploy an AI-based footfall detection solution using Azure and Azure Stack Hub

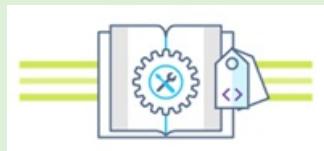
3/10/2022 • 7 minutes to read • [Edit Online](#)

This article describes how to deploy an AI-based solution that generates insights from real world actions by using Azure, Azure Stack Hub, and the Custom Vision AI Dev Kit.

In this solution, you learn how to:

- Deploy Cloud Native Application Bundles (CNAB) at the edge.
- Deploy an app that spans cloud boundaries.
- Use the Custom Vision AI Dev Kit for inference at the edge.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that allows you to build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Prerequisites

Before getting started with this deployment guide, make sure you:

- Review the [Footfall detection pattern](#) topic.
- Obtain user access to an Azure Stack Development Kit (ASDK) or Azure Stack Hub integrated system instance, with:
 - The [Azure App Service on Azure Stack Hub resource provider](#) installed. You need operator access to your Azure Stack Hub instance, or work with your administrator to install.
 - A subscription to an offer that provides App Service and Storage quota. You need operator access to create an offer.
- Obtain access to an Azure subscription.
 - If you don't have an Azure subscription, sign up for a [free trial account](#) before you begin.
- Create two service principals in your directory:
 - One set up for use with Azure resources, with access at the Azure subscription scope.
 - One set up for use with Azure Stack Hub resources, with access at the Azure Stack Hub subscription scope.
 - To learn more about creating service principals and authorizing access, see [Use an app identity to access resources](#). If you prefer to use Azure CLI, see [Create an Azure service principal with Azure CLI](#).
- Deploy Azure Cognitive Services in Azure or Azure Stack Hub.
 - First, [learn more about Cognitive Services](#).
 - Then visit [Deploy Azure Cognitive Services to Azure Stack Hub](#) to deploy Cognitive Services on Azure

Stack Hub. You first need to sign up for access to the preview.

- Clone or download an unconfigured Azure Custom Vision AI Dev Kit. For details, see the [Vision AI DevKit](#).
- Sign up for a Power BI account.
- An Azure Cognitive Services Face API subscription key and endpoint URL. You can get both with the [Try Cognitive Services](#) free trial. Or, follow the instructions in [Create a Cognitive Services account](#).
- Install the following development resources:
 - [Azure CLI 2.0](#)
 - [Docker CE](#)
 - [Porter](#). You use Porter to deploy cloud apps using CNAB bundle manifests that are provided for you.
 - [Visual Studio Code](#)
 - [Azure IoT Tools for Visual Studio Code](#)
 - [Python extension for Visual Studio Code](#)
 - [Python](#)

Deploy the hybrid cloud app

First you use the Porter CLI to generate a credential set, then deploy the cloud app.

1. Clone or download the [repository containing the solution sample code](#).
2. Porter will generate a set of credentials that will automate deployment of the app. Before running the credential generation command, be sure to have the following available:
 - A service principal for accessing Azure resources, including the service principal ID, key, and tenant DNS.
 - The subscription ID for your Azure subscription.
 - A service principal for accessing Azure Stack Hub resources, including the service principal ID, key, and tenant DNS.
 - The subscription ID for your Azure Stack Hub subscription.
 - Your Azure Cognitive Services Face API key and resource endpoint URL.
3. Run the Porter credential generation process and follow the prompts:

```
porter creds generate --tag intelligentedge/footfall-cloud-deployment:0.1.0
```

4. Porter also requires a set of parameters to run. Create a parameter text file and enter the following name/value pairs. Ask your Azure Stack Hub administrator if you need assistance with any of the required values.

NOTE

The `resource suffix` value is used to ensure that your deployment's resources have unique names across Azure. It must be a unique string of letters and numbers, no longer than 8 characters.

```
azure_stack_tenant_arm="Your Azure Stack Hub tenant endpoint"
azure_stack_storage_suffix="Your Azure Stack Hub storage suffix"
azure_stack_keyvault_suffix="Your Azure Stack Hub keyVault suffix"
resource_suffix="A unique string to identify your deployment"
azure_location="A valid Azure region"
azure_stack_location="Your Azure Stack Hub location identifier"
powerbi_display_name="Your first and last name"
powerbi_principal_name="Your Power BI account email address"
```

Save the text file and make a note of its path.

5. You're now ready to deploy the hybrid cloud app using Porter. Run the install command and watch as resources are deployed to Azure and Azure Stack Hub:

```
porter install footfall-cloud --tag intelligentedge/footfall-cloud-deployment:0.1.0 --creds footfall-cloud-deployment --param-file "path-to-cloud-parameters-file.txt"
```

6. Once deployment is complete, make note of the following values:

- The camera's connection string.
- The image storage account connection string.
- The resource group names.

Prepare the Custom Vision AI DevKit

Next, set up the Custom Vision AI Dev Kit as shown in the [Vision AI DevKit quickstart](#). You also set up and test your camera, using the connection string provided in the previous step.

Deploy the camera app

Use the Porter CLI to generate a credential set, then deploy the camera app.

1. Porter will generate a set of credentials that will automate deployment of the app. Before running the credential generation command, be sure to have the following available:
 - A service principal for accessing Azure resources, including the service principal ID, key, and tenant DNS.
 - The subscription ID for your Azure subscription.
 - The image storage account connection string provided when you deployed the cloud app.
2. Run the Porter credential generation process and follow the prompts:

```
porter creds generate --tag intelligentedge/footfall-camera-deployment:0.1.0
```

3. Porter also requires a set of parameters to run. Create a parameter text file and enter the following text. Ask your Azure Stack Hub administrator if you don't know some of the required values.

NOTE

The `deployment suffix` value is used to ensure that your deployment's resources have unique names across Azure. It must be a unique string of letters and numbers, no longer than 8 characters.

```
iot_hub_name="Name of the IoT Hub deployed"  
deployment_suffix="Unique string here"
```

Save the text file and make a note of its path.

4. You're now ready to deploy the camera app using Porter. Run the install command and watch as the IoT Edge deployment is created.

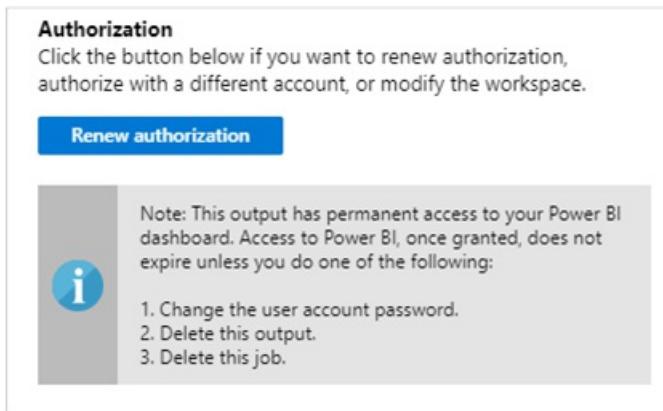
```
porter install footfall-camera --tag intelligentedge/footfall-camera-deployment:0.1.0 --creds footfall-camera-deployment --param-file "path-to-camera-parameters-file.txt"
```

- Verify that the camera's deployment is complete by viewing the camera feed at <https://<camera-ip>:3000/>, where <camera-ip> is the camera IP address. This step may take up to 10 minutes.

Configure Azure Stream Analytics

Now that data is flowing to Azure Stream Analytics from the camera, we need to manually authorize it to communicate with Power BI.

- From the Azure portal, open **All Resources**, and the *process-footfall[yoursuffix]* job.
- In the **Job Topology** section of the Stream Analytics job pane, select the **Outputs** option.
- Select the **traffic-output** output sink.
- Select **Renew authorization** and sign in to your Power BI account.



- Save the output settings.
- Go to the **Overview** pane and select **Start** to start sending data to Power BI.
- Select **Now** for job output start time and select **Start**. You can view the job status in the notification bar.

Create a Power BI Dashboard

- Once the job succeeds, go to [Power BI](#) and sign in with your work or school account. If the Stream Analytics job query is outputting results, the *footfall-dataset* dataset you created exists under the **Datasets** tab.
- From your Power BI workspace, select **+ Create** to create a new dashboard named *Footfall Analysis*.
- At the top of the window, select **Add tile**. Then select **Custom Streaming Data** and **Next**. Choose the *footfall-dataset* under **Your Datasets**. Select **Card** from the **Visualization type** dropdown, and add **age** to **Fields**. Select **Next** to enter a name for the tile, and then select **Apply** to create the tile.
- You can add additional fields and cards as desired.

Test Your Solution

Observe how the data in the cards you created in Power BI changes as different people walk in front of the camera. Inferences may take up to 20 seconds to appear once recorded.

Remove Your Solution

If you'd like to remove your solution, run the following commands using Porter, using the same parameter files that you created for deployment:

```
porter uninstall footfall-cloud -tag intelligenedge/footfall-cloud-deployment:0.1.0 -creds footfall-cloud-deployment -param-file "path-to-cloud-parameters-file.txt"
```

```
porter uninstall footfall-camera -tag intelligenedge/footfall-camera-deployment:0.1.0 -creds footfall-camera-deployment -param-file "path-to-camera-parameters-file.txt"
```

Next steps

- Learn more about [Hybrid app design considerations](#)
- Review and propose improvements to [the code for this sample on GitHub](#).

Deploy an app that scales cross-cloud using Azure and Azure Stack Hub

3/10/2022 • 12 minutes to read • [Edit Online](#)

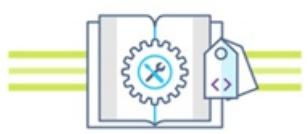
Learn how to create a cross-cloud solution to provide a manually triggered process for switching from an Azure Stack Hub hosted web app to an Azure hosted web app with autoscaling via traffic manager. This process ensures flexible and scalable cloud utility when under load.

With this pattern, your tenant may not be ready to run your app in the public cloud. However, it may not be economically feasible for the business to maintain the capacity required in their on-premises environment to handle spikes in demand for the app. Your tenant can make use of the elasticity of the public cloud with their on-premises solution.

In this solution, you'll build a sample environment to:

- Create a multi-node web app.
- Configure and manage the Continuous Deployment (CD) process.
- Publish the web app to Azure Stack Hub.
- Create a release.
- Learn to monitor and track your deployments.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that lets you build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Prerequisites

- Azure subscription. If needed, create a [free account](#) before beginning.
- An Azure Stack Hub integrated system or deployment of Azure Stack Development Kit (ASDK).
 - For instructions on installing Azure Stack Hub, see [Install the ASDK](#).
 - For an ASDK post-deployment automation script, go to: <https://github.com/mattmcspirit/azurestack>
 - This installation may require a few hours to complete.
- Deploy [App Service](#) PaaS services to Azure Stack Hub.
- [Create plans/offers](#) within the Azure Stack Hub environment.
- [Create tenant subscription](#) within the Azure Stack Hub environment.
- Create a web app within the tenant subscription. Make note of the new web app URL for later use.
- Deploy Azure Pipelines virtual machine (VM) within the tenant subscription.
- Windows Server 2016 VM with .NET 3.5 is required. This VM will be built in the tenant subscription on Azure Stack Hub as the private build agent.

- [Windows Server 2016 with SQL 2017 VM image](#) is available in the Azure Stack Hub Marketplace. If this image isn't available, work with an Azure Stack Hub Operator to ensure it's added to the environment.

Issues and considerations

Scalability

The key component of cross-cloud scaling is the ability to deliver immediate and on-demand scaling between public and on-premises cloud infrastructure, providing consistent and reliable service.

Availability

Ensure locally deployed apps are configured for high-availability through on-premises hardware configuration and software deployment.

Manageability

The cross-cloud solution ensures seamless management and familiar interface between environments.

PowerShell is recommended for cross-platform management.

Cross-cloud scaling

Get a custom domain and configure DNS

Update the DNS zone file for the domain. Azure AD will verify ownership of the custom domain name. Use [Azure DNS](#) for Azure/Microsoft 365/external DNS records within Azure, or add the DNS entry at [a different DNS registrar](#).

1. Register a custom domain with a public registrar.
2. Sign in to the domain name registrar for the domain. An approved admin may be required to make DNS updates.
3. Update the DNS zone file for the domain by adding the DNS entry provided by Azure AD. (The DNS entry won't affect email routing or web hosting behaviors.)

Create a default multi-node web app in Azure Stack Hub

Set up hybrid continuous integration and continuous deployment (CI/CD) to deploy web apps to Azure and Azure Stack Hub and to autopush changes to both clouds.

NOTE

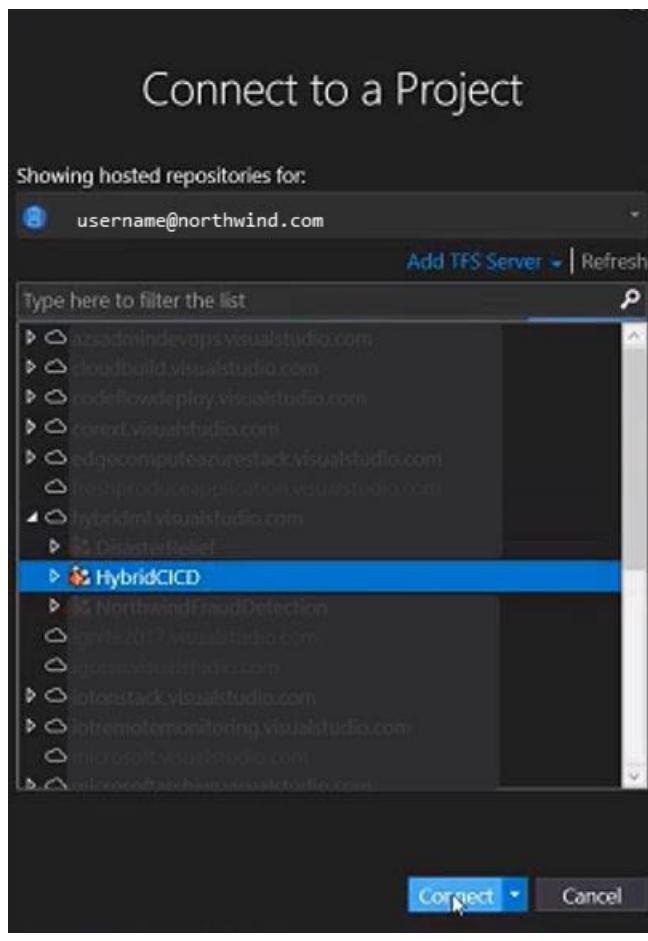
Azure Stack Hub with proper images syndicated to run (Windows Server and SQL) and App Service deployment are required. For more information, review the App Service documentation [Prerequisites for deploying App Service on Azure Stack Hub](#).

Add Code to Azure Repos

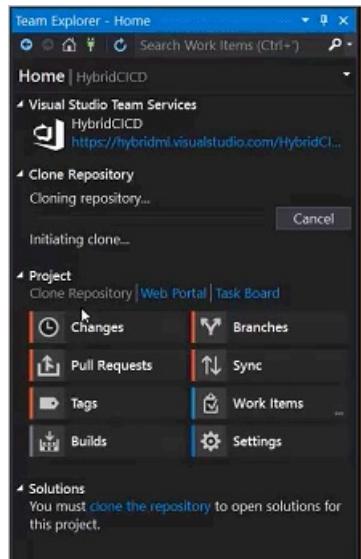
Azure Repos

1. Sign in to Azure Repos with an account that has project creation rights on Azure Repos.

Hybrid CI/CD can apply to both app code and infrastructure code. Use [Azure Resource Manager templates](#) for both private and hosted cloud development.



2. Clone the repository by creating and opening the default web app.



Create self-contained web app deployment for App Services in both clouds

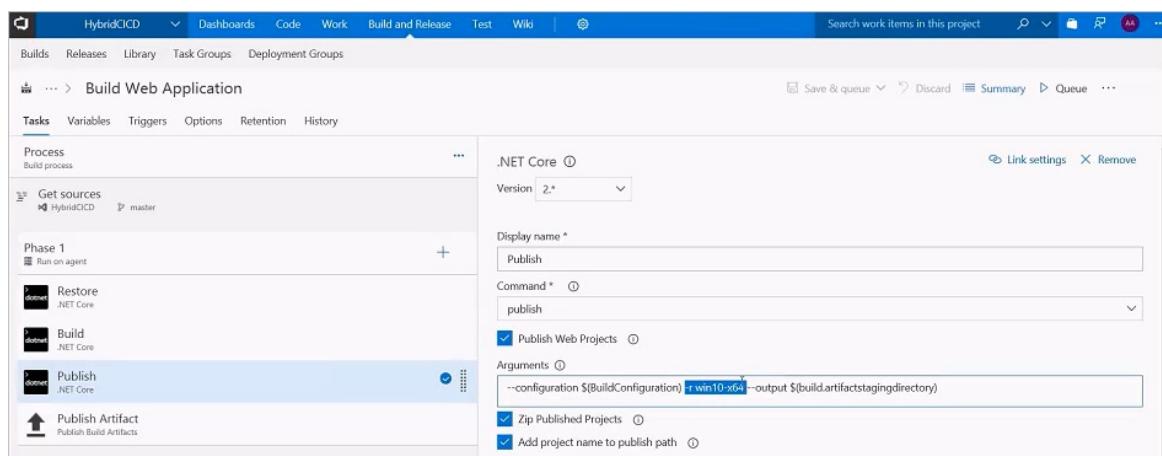
1. Edit the `WebApplication.csproj` file. Select `RuntimeIdentifier` and add `win10-x64`. (See [Self-contained deployment](#) documentation.)

```
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2   <PropertyGroup>
3     <TargetFramework>netcoreapp2.0</TargetFramework>
4     <RuntimeIdentifiers>win10-x64</RuntimeIdentifiers>
5   </PropertyGroup>
6   <ItemGroup>
7     <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.3" />
8   </ItemGroup>
9   <ItemGroup>
10    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.1" />
11  </ItemGroup>
12 </Project>
13
```

2. Check in the code to Azure Repos using Team Explorer.
3. Confirm that the app code has been checked into Azure Repos.

Create the build definition

1. Sign in to Azure Pipelines to confirm the ability to create build definitions.
2. Add -r win10-x64 code. This addition is necessary to trigger a self-contained deployment with .NET Core.



3. Run the build. The [self-contained deployment build](#) process will publish artifacts that run on Azure and Azure Stack Hub.

Use an Azure hosted agent

Using a hosted build agent in Azure Pipelines is a convenient option to build and deploy web apps. Maintenance and upgrades are done automatically by Microsoft Azure, enabling a continuous and uninterrupted development cycle.

Manage and configure the CD process

Azure Pipelines and Azure DevOps Services provide a highly configurable and manageable pipeline for releases to multiple environments like development, staging, QA, and production environments; including requiring approvals at specific stages.

Create release definition

1. Select the plus button to add a new release under the **Releases** tab in the **Build and Release** section of Azure DevOps Services.

The screenshot shows the 'Releases' tab selected in the top navigation bar. A context menu is open over a button, with 'Create release definition' highlighted. The main area displays 'All release definitions' with tabs for 'Overview', 'Releases', and 'Deleted'. Below is a table with columns for Title, Release Definition, and Environment.

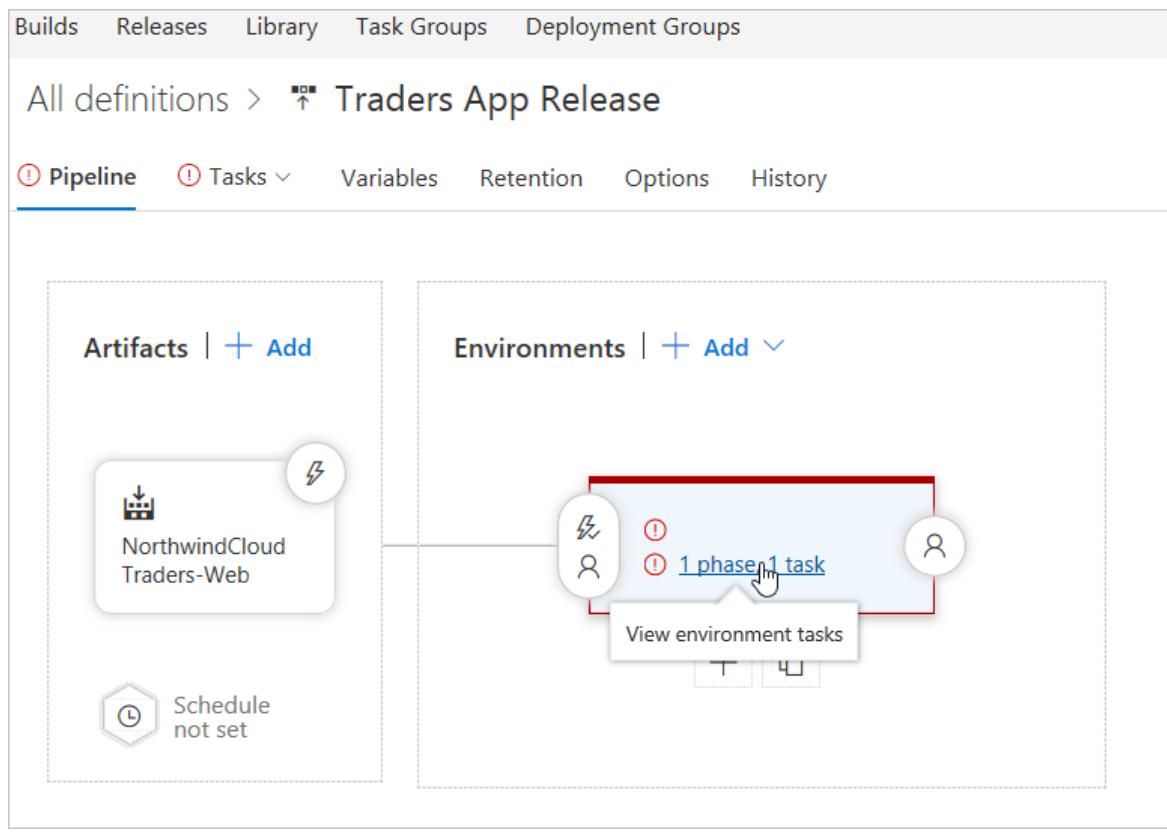
2. Apply the Azure App Service Deployment template.

The screenshot shows the 'Select a Template' dialog. It includes a search bar and a 'Featured' section with several deployment templates: 'Azure App Service Deployment' (selected), 'Deploy Node.js App to Azure App Service', 'Deploy PHP App to Azure App Service', and 'IIS Website and SQL Database Deployment'. The 'Apply' button is highlighted with a mouse cursor.

3. Under Add artifact, add the artifact for the Azure Cloud build app.

The screenshot shows the 'Add artifact' dialog. It lists 'Source type' options: 'Build' (selected), 'Git', 'GitHub', and 'Team Found...'. The 'Project' dropdown is set to 'NorthwindCloud'. The 'Source (Build definition)' dropdown shows a list of build definitions: 'NorthwindCloud Traders-Vessel', 'NorthwindCloud Traders-Web' (highlighted with a mouse cursor), 'NorthwindCloud-ModernizationApp', 'NorthwindCloud-PortTracker FI', and 'NorthwindCloud-PortTracker UY'.

4. Under Pipeline tab, select the Phase, Task link of the environment and set the Azure cloud environment values.



5. Set the **environment name** and select the **Azure subscription** for the Azure Cloud endpoint.

The screenshot shows the 'Tasks' tab for the 'Traders App Release' pipeline. Under the 'Azure' task, the 'Environment name' is set to 'Azure'. In the 'Parameters' section, the 'Azure subscription' dropdown is open, showing a list of available connections: 'AzureStack Traders-Vessel EP', 'AzureStack PortTracker UY EP', 'AzureStack PortTracker FI EP', 'AzureCloud RegulatoryApp EP', 'AzureStack ModernizationApp Inventory EP', and 'AzureCloud Traders-Web EP'. The 'AzureCloud Traders-Web EP' option is highlighted with a cursor. The top navigation bar includes 'Builds', 'Releases', 'Library', 'Task Groups', and 'Deployment Groups'. The 'Save' button is visible at the top right.

6. Under **App service name**, set the required Azure app service name.

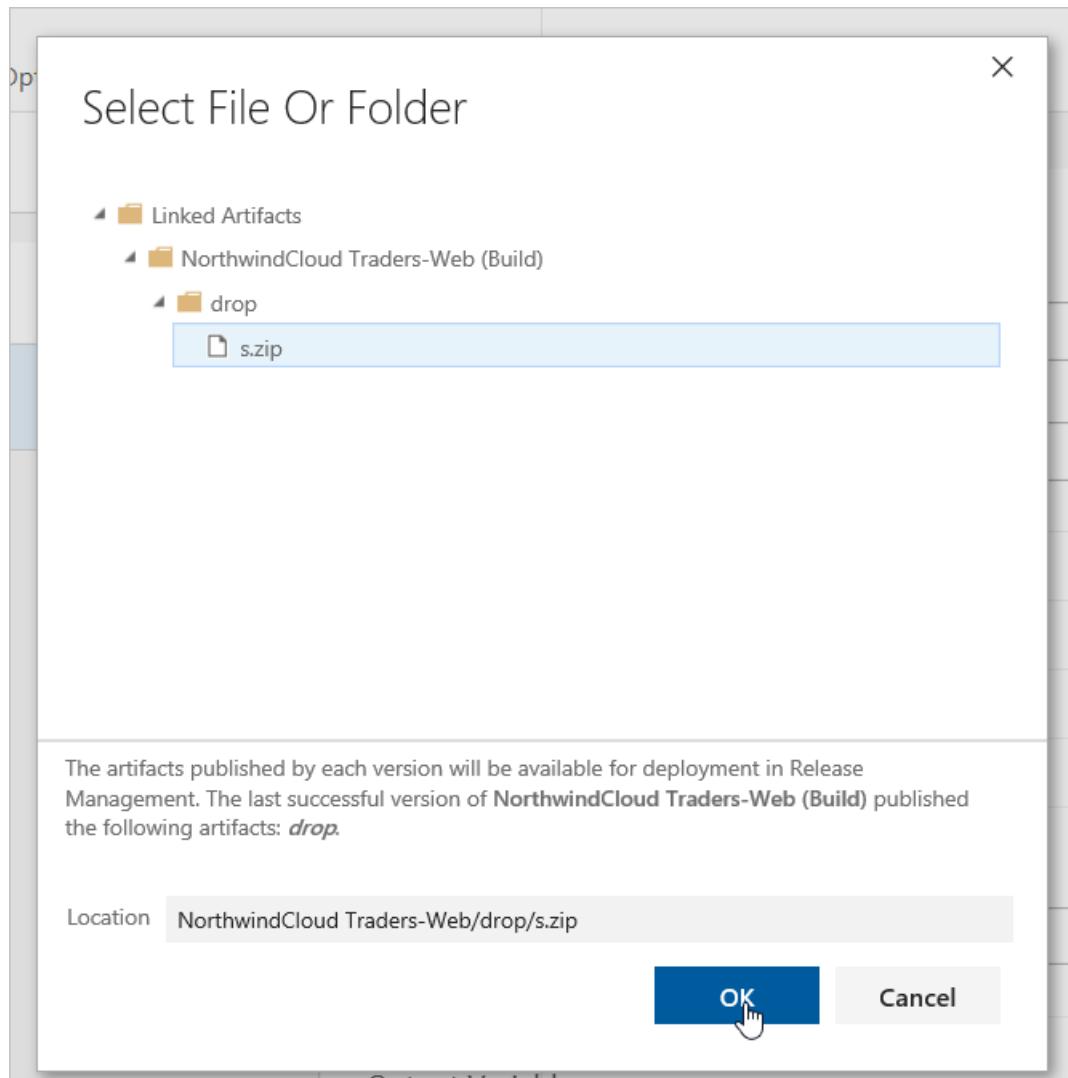
The screenshot shows the 'Tasks' tab for the 'Traders App Release' pipeline. Under the 'Azure' task, the 'Environment name' is set to 'Azure'. In the 'Parameters' section, the 'Azure subscription' dropdown is set to 'AzureCloud Traders-Web EP'. Below it, the 'App type' is set to 'Web App'. In the 'App service name' dropdown, the value 'northwindtraders' is selected. A note below the field states: 'This field is linked to 1 setting in 'Deploy Azure App Service''. The top navigation bar includes 'Builds', 'Releases', 'Library', 'Task Groups', and 'Deployment Groups'. The 'Save' button is visible at the top right.

7. Enter "Hosted VS2017" under **Agent queue** for Azure cloud hosted environment.

The screenshot shows the 'Traders App Release' pipeline configuration. In the 'Tasks' tab, a 'Run on agent' step is selected. On the right, the 'Agent phase' configuration is shown, with the 'Agent queue' dropdown set to 'Hosted VS2017'. A modal window displays a list of agent queues, with 'Hosted VS2017' highlighted.

8. In Deploy Azure App Service menu, select the valid **Package or Folder** for the environment. Select **OK** to **folder location**.

The screenshot shows the 'Traders App Release' pipeline configuration. In the 'Tasks' tab, a 'Deploy Azure App Service' step is selected. On the right, the 'Virtual application' and 'Package or folder' fields are configured. The 'Package or folder' field contains the value '\$(System.DefaultWorkingDirectory)/**/*.zip'.



9. Save all changes and go back to release pipeline.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Deployment process

Run on agent Run on agent

Deploy Azure App Service Azure App Service Deploy

Environment name: Azure

Parameters: Unlink all

Azure subscription: AzureCloud Traders-Web EP

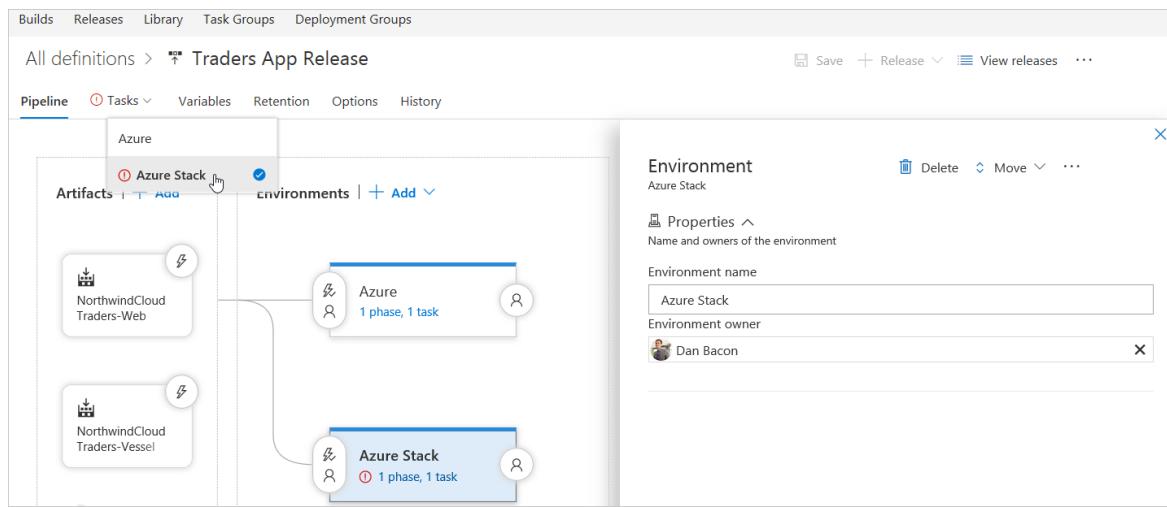
App type: Web App

10. Add a new artifact selecting the build for the Azure Stack Hub app.

11. Add one more environment by applying the Azure App Service Deployment.

12. Name the new environment "Azure Stack".

13. Find the Azure Stack environment under Task tab.



14. Select the subscription for the Azure Stack endpoint.

15. Set the Azure Stack web app name as the App service name.

16. Select the Azure Stack agent.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Stack Deployment process

Run on agent Run on agent

Deploy Azure App Service Azure App Service Deploy

Agent phase Agent selection Agent queue

Display name * Run on agent

AzureStack - Douglas Fir Hosted Hosted Hosted Linux Preview Hosted macOS Preview Hosted VS2017 Private AzureStack - Douglas Fir Default (No agents)

Save Release View releases ...

17. Under the Deploy Azure App Service section, select the valid **Package or Folder** for the environment. Select **OK** to folder location.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Stack Deployment process

Run on agent Run on agent

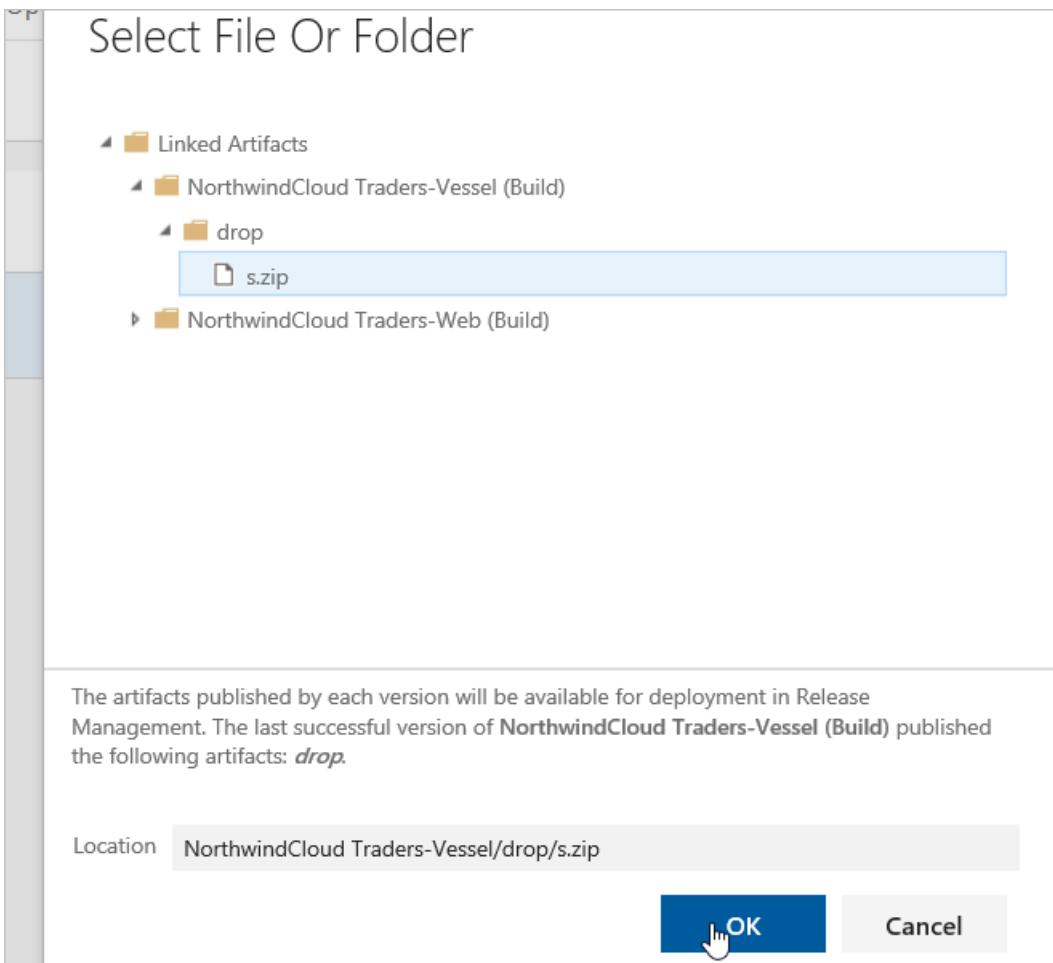
Deploy Azure App Service Azure App Service Deploy

Virtual application Deploy to slot

Package or folder \$System.DefaultWorkingDirectory/**/*.zip

File Transforms & Variable Substitution Options Additional Deployment Options Post Deployment Action Application and Configuration Settings Output App Service URL

Save Release View releases ...

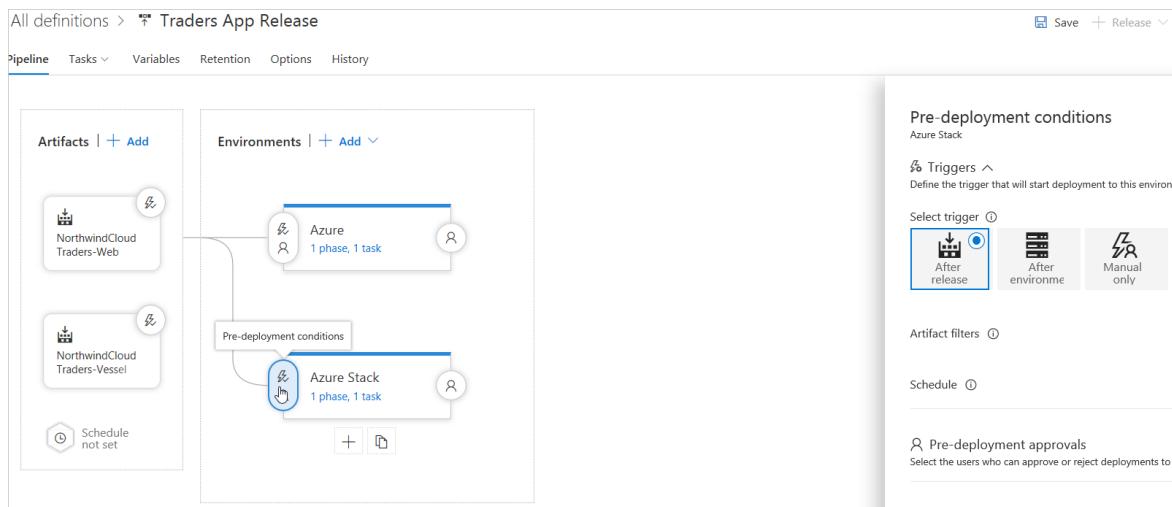


18. Under Variable tab add a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS`, set its value as `true`, and scope to Azure Stack.

Name	Value	Scope
<code>VSTS_ARM_REST_IGNORE_SSL_ERRORS</code>	true	Azure Stack

19. Select the **Continuous** deployment trigger icon in both artifacts and enable the **Continuous** deployment trigger.

20. Select the **Pre-deployment** conditions icon in the Azure Stack environment and set the trigger to **After release**.



21. Save all changes.

NOTE

Some settings for the tasks may have been automatically defined as [environment variables](#) when creating a release definition from a template. These settings can't be modified in the task settings; instead, the parent environment item must be selected to edit these settings.

Publish to Azure Stack Hub via Visual Studio

By creating endpoints, an Azure DevOps Services build can deploy Azure Service apps to Azure Stack Hub. Azure Pipelines connects to the build agent, which connects to Azure Stack Hub.

1. Sign in to Azure DevOps Services and go to the app settings page.
2. On **Settings**, select **Security**.
3. In **VSTS Groups**, select **Endpoint Creators**.
4. On the **Members** tab, select **Add**.
5. In **Add users and groups**, enter a user name and select that user from the list of users.
6. Select **Save changes**.
7. In the **VSTS Groups** list, select **Endpoint Administrators**.
8. On the **Members** tab, select **Add**.
9. In **Add users and groups**, enter a user name and select that user from the list of users.
10. Select **Save changes**.

Now that the endpoint information exists, the Azure Pipelines to Azure Stack Hub connection is ready to use. The build agent in Azure Stack Hub gets instructions from Azure Pipelines and then the agent conveys endpoint information for communication with Azure Stack Hub.

Develop the app build

NOTE

Azure Stack Hub with proper images syndicated to run (Windows Server and SQL) and App Service deployment are required. For more information, see [Prerequisites for deploying App Service on Azure Stack Hub](#).

Use [Azure Resource Manager templates](#) like web app code from Azure Repos to deploy to both clouds.

Add code to an Azure Repos project

1. Sign in to Azure Repos with an account that has project creation rights on Azure Stack Hub.
2. **Clone the repository** by creating and opening the default web app.

Create self-contained web app deployment for App Services in both clouds

1. Edit the **WebApplication.csproj** file: Select **RuntimeIdentifier** and then add **win10-x64**. For more information, see [Self-contained deployment](#) documentation.

2. Use Team Explorer to check the code into Azure Repos.
3. Confirm that the app code was checked into Azure Repos.

Create the build definition

1. Sign in to Azure Pipelines with an account that can create a build definition.
2. Go to the **Build Web Application** page for the project.
3. In **Arguments**, add **-r win10-x64** code. This addition is required to trigger a self-contained deployment with .NET Core.
4. Run the build. The [self-contained deployment build](#) process will publish artifacts that can run on Azure and Azure Stack Hub.

Use an Azure hosted build agent

Using a hosted build agent in Azure Pipelines is a convenient option to build and deploy web apps. Maintenance and upgrades are done automatically by Microsoft Azure, enabling a continuous and uninterrupted development cycle.

Configure the continuous deployment (CD) process

Azure Pipelines and Azure DevOps Services provide a highly configurable and manageable pipeline for releases to multiple environments like development, staging, quality assurance (QA), and production. This process can include requiring approvals at specific stages of the app life cycle.

Create release definition

Creating a release definition is the final step in the app build process. This release definition is used to create a release and deploy a build.

1. Sign in to Azure Pipelines and go to **Build and Release** for the project.
2. On the **Releases** tab, select [+] and then pick **Create release definition**.
3. On **Select a Template**, choose **Azure App Service Deployment**, and then select **Apply**.
4. On **Add artifact**, from the **Source (Build definition)**, select the Azure Cloud build app.
5. On the **Pipeline** tab, select the **1 Phase, 1 Task** link to **View environment tasks**.
6. On the **Tasks** tab, enter Azure as the **Environment name** and select the AzureCloud Traders-Web EP from the **Azure subscription** list.
7. Enter the **Azure app service name**, which is **northwindtraders** in the next screen capture.

8. For the Agent phase, select **Hosted VS2017** from the **Agent queue** list.
9. In **Deploy Azure App Service**, select the valid **Package or folder** for the environment.
10. In **Select File or Folder**, select **OK** to **Location**.
11. Save all changes and go back to **Pipeline**.
12. On the **Pipeline** tab, select **Add artifact**, and choose the **NorthwindCloud Traders-Vessel** from the **Source (Build Definition)** list.
13. On **Select a Template**, add another environment. Pick **Azure App Service Deployment** and then select **Apply**.
14. Enter **Azure Stack Hub** as the **Environment name**.
15. On the **Tasks** tab, find and select **Azure Stack Hub**.
16. From the **Azure subscription** list, select **AzureStack Traders-Vessel EP** for the **Azure Stack Hub endpoint**.
17. Enter the **Azure Stack Hub** web app name as the **App service name**.
18. Under **Agent selection**, pick **AzureStack -b Douglas Fir** from the **Agent queue** list.
19. For **Deploy Azure App Service**, select the valid **Package or folder** for the environment. On **Select File Or Folder**, select **OK** for the folder **Location**.
20. On the **Variable** tab, find the variable named **VSTS_ARM_REST_IGNORE_SSL_ERRORS**. Set the variable value to **true**, and set its scope to **Azure Stack Hub**.
21. On the **Pipeline** tab, select the **Continuous deployment trigger** icon for the **NorthwindCloud Traders-Web** artifact and set the **Continuous deployment trigger** to **Enabled**. Do the same thing for the **NorthwindCloud Traders-Vessel** artifact.
22. For the **Azure Stack Hub** environment, select the **Pre-deployment conditions** icon set the trigger to **After release**.
23. Save all changes.

NOTE

Some settings for release tasks are automatically defined as [environment variables](#) when creating a release definition from a template. These settings can't be modified in the task settings but can be modified in the parent environment items.

Create a release

1. On the **Pipeline** tab, open the **Release** list and select **Create release**.
2. Enter a description for the release, check to see that the correct artifacts are selected, and then select **Create**. After a few moments, a banner appears indicating that the new release was created and the release name is displayed as a link. Select the link to see the release summary page.
3. The release summary page shows details about the release. In the following screen capture for "Release-2", the **Environments** section shows the **Deployment status** for Azure as "IN PROGRESS", and the status for Azure Stack Hub is "SUCCEEDED". When the deployment status for the Azure environment changes to "SUCCEEDED", a banner appears indicating that the release is ready for approval. When a deployment is pending or has failed, a blue (i) information icon is shown. Hover over the icon to see a pop-up that contains the reason for delay or failure.

4. Other views, like the list of releases, also display an icon that indicates approval is pending. The pop-up for this icon shows the environment name and more details related to the deployment. It's easy for an admin see the overall progress of releases and see which releases are waiting for approval.

Monitor and track deployments

1. On the **Release-2** summary page, select **Logs**. During a deployment, this page shows the live log from the agent. The left pane shows the status of each operation in the deployment for each environment.
2. Select the person icon in the **Action** column for a pre-deployment or post-deployment approval to see who approved (or rejected) the deployment and the message they provided.
3. After the deployment finishes, the entire log file is displayed in the right pane. Select any **Step** in the left pane to see the log file for a single step, like **Initialize Job**. The ability to see individual logs makes it easier to trace and debug parts of the overall deployment. **Save** the log file for a step or **Download all logs as zip**.
4. Open the **Summary** tab to see general information about the release. This view shows details about the build, the environments it was deployed to, deployment status, and other information about the release.
5. Select an environment link (**Azure** or **Azure Stack Hub**) to see information about existing and pending deployments to a specific environment. Use these views as a quick way to check that the same build was deployed to both environments.
6. Open the **deployed production app** in a browser. For example, for the Azure App Services website, open the URL `https://[your-app-name]\.azurewebsites.net`.

Integration of Azure and Azure Stack Hub provides a scalable cross-cloud solution

A flexible and robust multi-cloud service provides data security, back up and redundancy, consistent and rapid availability, scalable storage and distribution, and geo-compliant routing. This manually triggered process ensures reliable and efficient load switching between hosted web apps and immediate availability of crucial data.

Next steps

- To learn more about Azure Cloud Patterns, see [Cloud Design Patterns](#).

Deploy a high availability Kubernetes cluster on Azure Stack Hub

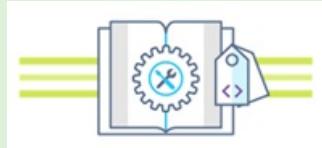
3/10/2022 • 11 minutes to read • [Edit Online](#)

This article will show you how to build a highly available Kubernetes cluster environment, deployed on multiple Azure Stack Hub instances, in different physical locations.

In this solution deployment guide, you learn how to:

- Download and prepare the AKS Engine
- Connect to the AKS Engine Helper VM
- Deploy a Kubernetes cluster
- Connect to the Kubernetes cluster
- Connect Azure Pipelines to Kubernetes cluster
- Configure monitoring
- Deploy application
- Autoscale application
- Configure Traffic Manager
- Upgrade Kubernetes
- Scale Kubernetes

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that allows you to build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Prerequisites

Before getting started with this deployment guide, make sure you:

- Review the [High availability Kubernetes cluster pattern](#) article.
- Review the contents of the [companion GitHub repository](#), which contains additional assets referenced in this article.
- Have an account that can access the [Azure Stack Hub user portal](#), with at least "contributor" permissions.

Download and prepare AKS Engine

AKS Engine is a binary that can be used from any Windows or Linux host that can reach the Azure Stack Hub Azure Resource Manager endpoints. This guide describes deploying a new Linux (or Windows) VM on Azure Stack Hub. It will be used later when AKS Engine deploys the Kubernetes clusters.

NOTE

You can also use an existing Windows or Linux VM to deploy a Kubernetes cluster on Azure Stack Hub using AKS Engine.

The step-by-step process and requirements for AKS Engine are documented here:

- [Install the AKS Engine on Linux in Azure Stack Hub](#) (or using [Windows](#))

AKS Engine is a helper tool to deploy and operate (unmanaged) Kubernetes clusters (in Azure and Azure Stack Hub).

The details and differences of AKS Engine on Azure Stack Hub are described here:

- [What is the AKS Engine on Azure Stack Hub?](#)
- [AKS Engine on Azure Stack Hub](#) (on GitHub)

The sample environment will use Terraform to automate the deployment of the AKS Engine VM. You can find the [details and code in the companion GitHub repo](#).

The result of this step is a new resource group on Azure Stack Hub that contains the AKS Engine helper VM and related resources:

<input type="checkbox"/> NAME ↑↓	TYPE ↑↓	LOCATION ↑↓
<input type="checkbox"/>  aksengine-nic1	Network interface	seattle
<input type="checkbox"/>  aksengine-nsg	Network security group	seattle
<input type="checkbox"/>  aksengine-pip	Public IP address	seattle
<input type="checkbox"/>  aksengine-vm1	Virtual machine	seattle
<input type="checkbox"/>  aksengine-vm1-osdisk	Disk	seattle
<input type="checkbox"/>  aksengine-vnet	Virtual network	seattle

NOTE

If you have to deploy AKS Engine in a disconnected air-gapped environment, review [Disconnected Azure Stack Hub Instances](#) to learn more.

In the next step, we'll use the newly deployed AKS Engine VM to deploy a Kubernetes cluster.

Connect to the AKS Engine helper VM

First you must connect to the previously created AKS Engine helper VM.

The VM should have a Public IP Address and should be accessible via SSH (Port 22/TCP).

Resource group aksengine	Computer name (not available)
Status Running	Operating system Linux
Location seattle	Size Standard F2 (2 vcpus, 4 GiB memory)
Subscription AKSeArch	Public IP address 199.6.94.80

TIP

You can use a tool of your choice like MobaXterm, puTTY or PowerShell in Windows 10 to connect to a Linux VM using SSH.

```
ssh <username>@<ipaddress>
```

After connecting, run the command `aks-engine`. Go to [Supported AKS Engine Versions](#) to learn more about the AKS Engine and Kubernetes versions.

```
azureadmin@topicalredbird:~$ aks-engine
```

Usage:

```
  aks-engine [flags]
  aks-engine [command]
```

Available Commands:

completion	Generates bash completion scripts
deploy	Deploy an Azure Resource Manager template
generate	Generate an Azure Resource Manager template
get-versions	Display info about supported Kubernetes versions
help	Help about any command
rotate-certs	Rotate certificates on an existing Kubernetes cluster
scale	Scale an existing Kubernetes cluster
upgrade	Upgrade an existing Kubernetes cluster
version	Print the version of AKS Engine

Flags:

--debug	enable verbose debug logs
-h, --help	help for aks-engine
--show-default-model	Dump the default API model to stdout

Use "aks-engine [command] --help" for more information about a command.

Deploy a Kubernetes cluster

The AKS Engine helper VM itself hasn't created a Kubernetes cluster on our Azure Stack Hub, yet. Creating the cluster is the first action to take in the AKS Engine helper VM.

The step-by-step process is documented here:

- [Deploy a Kubernetes cluster with the AKS engine on Azure Stack Hub](#)

The end result of the `aks-engine deploy` command and the preparations in the previous steps is a fully featured Kubernetes cluster deployed into the tenant space of the first Azure Stack Hub instance. The cluster itself consists of Azure IaaS components like VMs, load balancers, VNets, disks, and so on.

<input type="checkbox"/>	linuxpool-availabilitySet-35064155	Availability set
<input type="checkbox"/>	master-availabilityset-35064155	Availability set
<input type="checkbox"/>	k8s-linuxpool-35064155-0_OsDisk_1_c1856a4dc69...	Disk
<input type="checkbox"/>	k8s-linuxpool-35064155-1_OsDisk_1_8e7ebcd8aca...	Disk
<input type="checkbox"/>	k8s-linuxpool-35064155-2_OsDisk_1_22fda474b6a...	Disk
<input type="checkbox"/>	k8s-master-35064155-0_OsDisk_1_41f3a2ca50a148...	Disk
<input type="checkbox"/>	k8s-master-35064155-0-etcddisk	Disk
<input type="checkbox"/>	k8s-master-lb-35064155	Load balancer 1
<input type="checkbox"/>	k8s-linuxpool-35064155-nic-0	Network interface
<input type="checkbox"/>	k8s-linuxpool-35064155-nic-1	Network interface
<input type="checkbox"/>	k8s-linuxpool-35064155-nic-2	Network interface
<input type="checkbox"/>	k8s-master-35064155-nic-0	Network interface
<input type="checkbox"/>	k8s-master-35064155-nsg	Network security group
<input type="checkbox"/>	k8s-master-ip-aks-35064155	Public IP address
<input type="checkbox"/>	k8s-master-35064155-routetable	Route table
<input type="checkbox"/>	k8s-linuxpool-35064155-0	Virtual machine 2
<input type="checkbox"/>	k8s-linuxpool-35064155-1	Virtual machine
<input type="checkbox"/>	k8s-linuxpool-35064155-2	Virtual machine
<input type="checkbox"/>	k8s-master-35064155-0	Virtual machine 3
<input type="checkbox"/>	k8s-vnet-35064155	Virtual network

1. Azure load balancer (K8s API Endpoint)
- 2) Worker Nodes (Agent Pool)
- 3) Master Nodes

The cluster is now up-and-running and in the next step we'll connect to it.

Connect to the Kubernetes cluster

You can now connect to the previously created Kubernetes cluster, either via SSH (using the SSH key specified as part of the deployment) or via `kubectl` (recommended). The Kubernetes command-line tool `kubectl` is available for Windows, Linux, and macOS [here](#). It's already pre-installed and configured on the master nodes of our cluster:

```
ssh azureuser@<k8s-master-lb-ip>
```

```
azureuser@k8s-master-35064155-0:~$ kubectl
kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create      Create a resource from a file or from stdin.
  expose      Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
  run         Run a particular image on the cluster
  set         Set specific features on objects
```

It's not recommended to use the master node as a jumpbox for administrative tasks. The `kubectl` configuration is stored in `.kube/config` on the master node(s) as well as on the AKS Engine VM. You can copy the configuration to an admin machine with connectivity to the Kubernetes cluster and use the `kubectl` command there. The `.kube/config` file is also used later to configure a service connection in Azure Pipelines.

IMPORTANT

Keep these files secure because they contain the credentials for your Kubernetes cluster. An attacker with access to the file has enough information to gain administrator access to it. All actions that are done using the initial `.kube/config` file are done using a cluster-admin account.

You can now try various commands using `kubectl` to check the status of your cluster. Here are example commands:

```
kubectl get nodes
```

NAME	STATUS	ROLE	VERSION
k8s-linuxpool-35064155-0	Ready	agent	v1.14.8
k8s-linuxpool-35064155-1	Ready	agent	v1.14.8
k8s-linuxpool-35064155-2	Ready	agent	v1.14.8
k8s-master-35064155-0	Ready	master	v1.14.8

```
kubectl cluster-info
```

```
Kubernetes master is running at https://aks.***
CoreDNS is running at https://aks.***/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
kubernetes-dashboard is running at https://aks.***/api/v1/namespaces/kube-system/services/https:kubernetes-
dashboard:/proxy
Metrics-server is running at https://aks.***/api/v1/namespaces/kube-system/services/https:metrics-
server:/proxy
```

IMPORTANT

Kubernetes has its own **Role-based Access Control (RBAC)** model that allows you to create fine-grained role definitions and role bindings. This is the preferable way to control access to the cluster instead of handing out cluster-admin permissions.

Connect Azure Pipelines to Kubernetes clusters

To connect Azure Pipelines to the newly deployed Kubernetes cluster, we need its `kubeconfig` (`.kube/config`) file as explained in the previous step.

- Connect to one of the master nodes of your Kubernetes cluster.
- Copy the content of the `.kube/config` file.

- Go to Azure DevOps > Project Settings > Service Connections to create a new "Kubernetes" service connection (use KubeConfig as Authentication method)

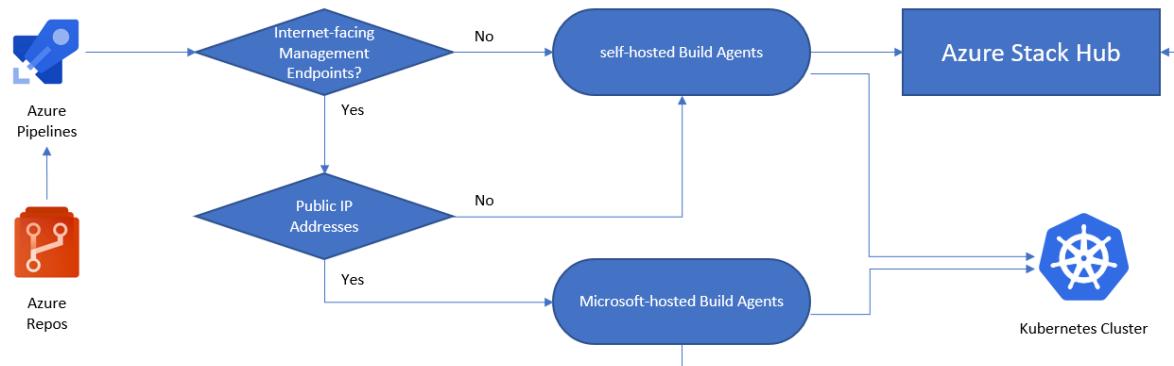
IMPORTANT

Azure Pipelines (or its build agents) must have access to the Kubernetes API. If there is an Internet connection from Azure Pipelines to the Azure Stack Hub Kubernetes cluster, you'll need to deploy a self-hosted Azure Pipelines Build Agent.

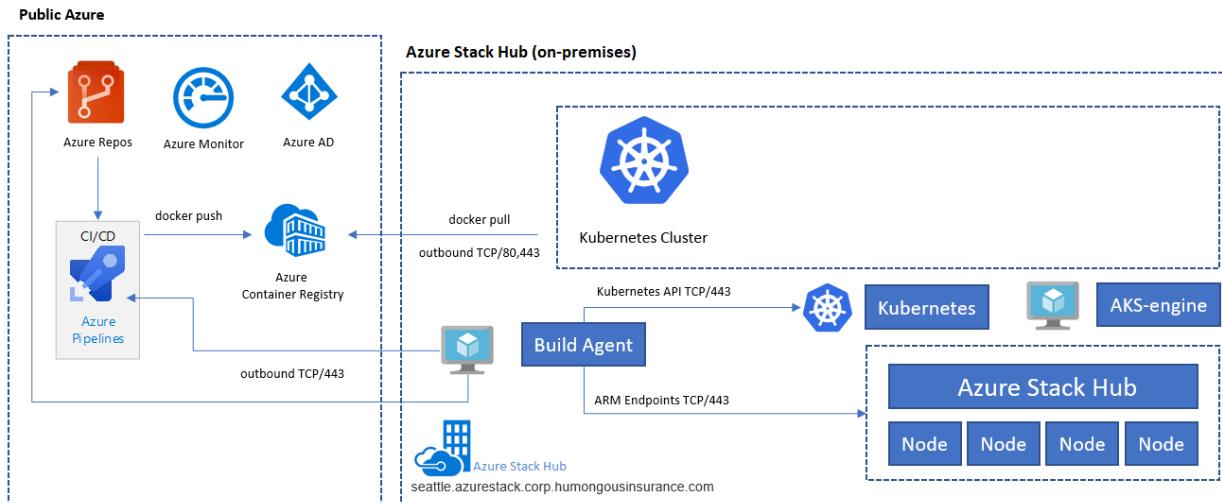
When deploying self-hosted Agents for Azure Pipelines, you may deploy either on Azure Stack Hub, or on a machine with network connectivity to all required management endpoints. See the details here:

- [Azure Pipelines agents on Windows or Linux](#)

The pattern [Deployment \(CI/CD\) considerations](#) section contains a decision flow that helps you to understand whether to use Microsoft-hosted agents or self-hosted agents:



In this sample solution, the topology includes a self-hosted build agent on each Azure Stack Hub instance. The agent can access the Azure Stack Hub Management Endpoints and the Kubernetes cluster API endpoints.



This design fulfills a common regulatory requirement, which is to have only outbound connections from the application solution.

Configure monitoring

You can use [Azure Monitor](#) for containers to monitor the containers in the solution. This points Azure Monitor to the AKS Engine-deployed Kubernetes cluster on Azure Stack Hub.

There are two ways to enable Azure Monitor on your cluster. Both ways require you to set up a Log Analytics

workspace in Azure.

- [Method one](#) uses a Helm Chart
- [Method two](#) as part of the AKS Engine cluster specification

In the sample topology, "Method one" is used, which allows automation of the process and updates can be installed more easily.

For the next step, you need an Azure LogAnalytics Workspace (ID and Key), `Helm` (version 3), and `kubectl` on your machine.

Helm is a Kubernetes package manager, available as a binary that is runs on macOS, Windows, and Linux. It can be downloaded at [helm.sh](#). Helm relies on the Kubernetes configuration file used for the `kubectl` command:

```
helm repo add incubator https://kubernetes-charts-incubator.storage.googleapis.com/
helm repo update

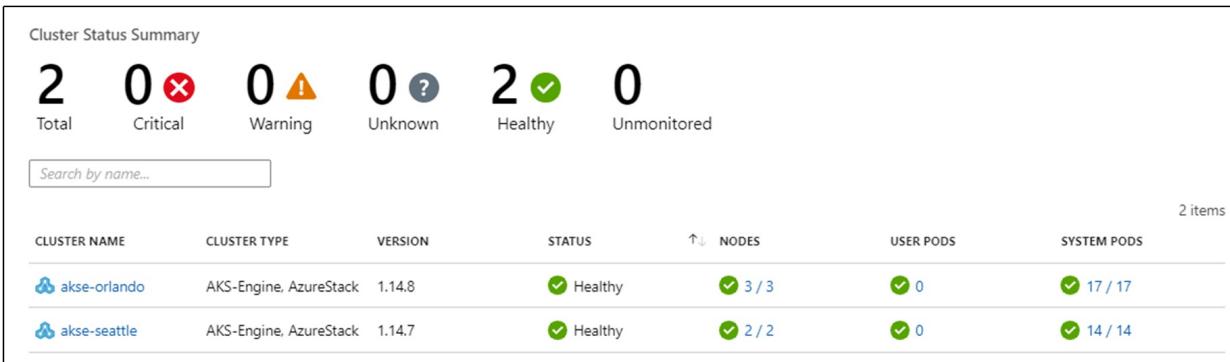
helm install incubator/azuremonitor-containers \
--set omsagent.secret.wsid=<your_workspace_id> \
--set omsagent.secret.key=<your_workspace_key> \
--set omsagent.env.clusterName=<my_prod_cluster> \
--generate-name
```

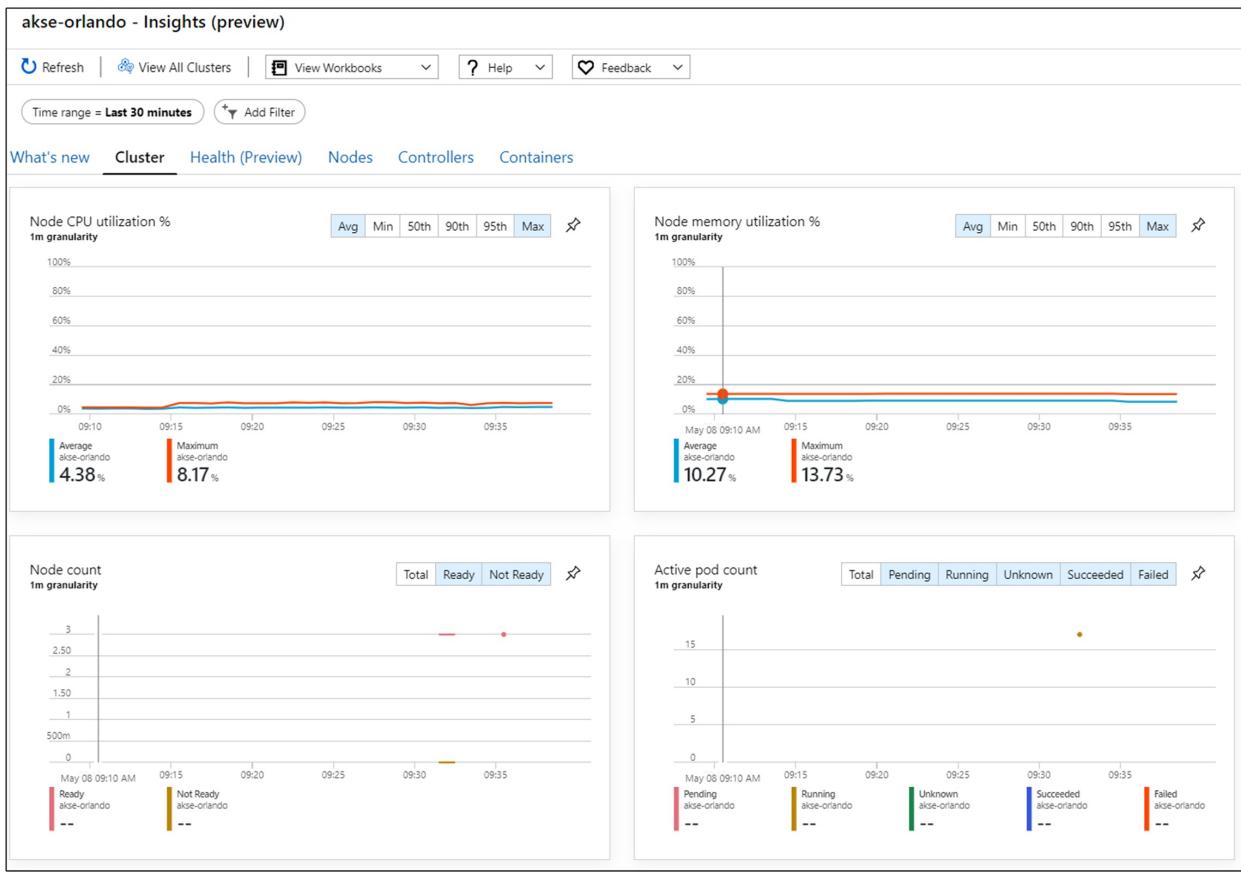
This command will install the Azure Monitor agent on your Kubernetes cluster:

```
kubectl get pods -n kube-system
```

NAME	READY	STATUS
omsagent-8qdm6	1/1	Running
omsagent-r6ppm	1/1	Running
omsagent-rs-76c45758f5-lmc4l	1/1	Running

The Operations Management Suite (OMS) Agent on your Kubernetes cluster will send monitoring data to your Azure Log Analytics Workspace (using outbound HTTPS). You can now use Azure Monitor to get deeper insights about your Kubernetes clusters on Azure Stack Hub. This design is a powerful way to demonstrate the power of analytics that can be automatically deployed with your application's clusters.





IMPORTANT

If Azure Monitor does not show any Azure Stack Hub data, please make sure that you have followed the instructions on [how to add AzureMonitor-Containers solution to a Azure Log Analytics workspace](#) carefully.

Deploy the application

Before installing our sample application, there's another step to configure the nginx-based Ingress controller on our Kubernetes cluster. The Ingress controller is used as a layer 7 load balancer to route traffic in our cluster based on host, path, or protocol. Nginx-ingress is available as a Helm Chart. For detailed instructions, refer to the [Helm Chart GitHub repository](#).

Our sample application is also packaged as a Helm Chart, like the [Azure Monitoring Agent](#) in the previous step. As such, it's straightforward to deploy the application onto our Kubernetes cluster. You can find the [Helm Chart files in the companion GitHub repo](#)

The sample application is a three tier application, deployed onto a Kubernetes cluster on each of two Azure Stack Hub instances. The application uses a MongoDB database. You can learn more about how to get the data replicated across multiple instances in the pattern [Data and Storage considerations](#).

After deploying the Helm Chart for the application, you'll see all three tiers of your application represented as deployments and stateful sets (for the database) with a single pod:

```
kubectl get pod,deployment,statefulset
```

NAME	READY	STATUS
pod/ratings-api-569d7f7b54-mrv5d	1/1	Running
pod/ratings-mongodb-0	1/1	Running
pod/ratings-web-85667bfb86-16vxz	1/1	Running
NAME	READY	
deployment.extensions/ratings-api	1/1	
deployment.extensions/ratings-web	1/1	
NAME	READY	
statefulset.apps/ratings-mongodb	1/1	

On the services, side you'll find the nginx-based Ingress Controller and its public IP address:

```
kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP
nginx-ingress-1588931383-controller	LoadBalancer	10.0.114.180	*public-ip*	443:30667/TCP
nginx-ingress-1588931383-default-backend	ClusterIP	10.0.76.54	<none>	80/TCP
ratings-api	ClusterIP	10.0.46.69	<none>	80/TCP
ratings-web	ClusterIP	10.0.161.124	<none>	80/TCP

The "External IP" address is our "application endpoint". It's how users will connect to open the application and will also be used as the endpoint for our next step [Configure Traffic Manager](#).

Autoscale the application

You can optionally configure the [Horizontal Pod Autoscaler](#) to scale up or down based on certain metrics like CPU utilization. The following command will create a Horizontal Pod Autoscaler that maintains 1 to 10 replicas of the Pods controlled by the ratings-web deployment. HPA will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all Pods of 80%:

```
kubectl autoscale deployment ratings-web --cpu-percent=80 --min=1 --max=10
```

You may check the current status of autoscaler by running this command:

```
kubectl get hpa
```

NAME	REFERENCE	TARGET	MINPODS	MAXPODS	REPLICAS	AGE
ratings-web	Deployment/ratings-web/scale	0% / 80%	1	10	1	18s

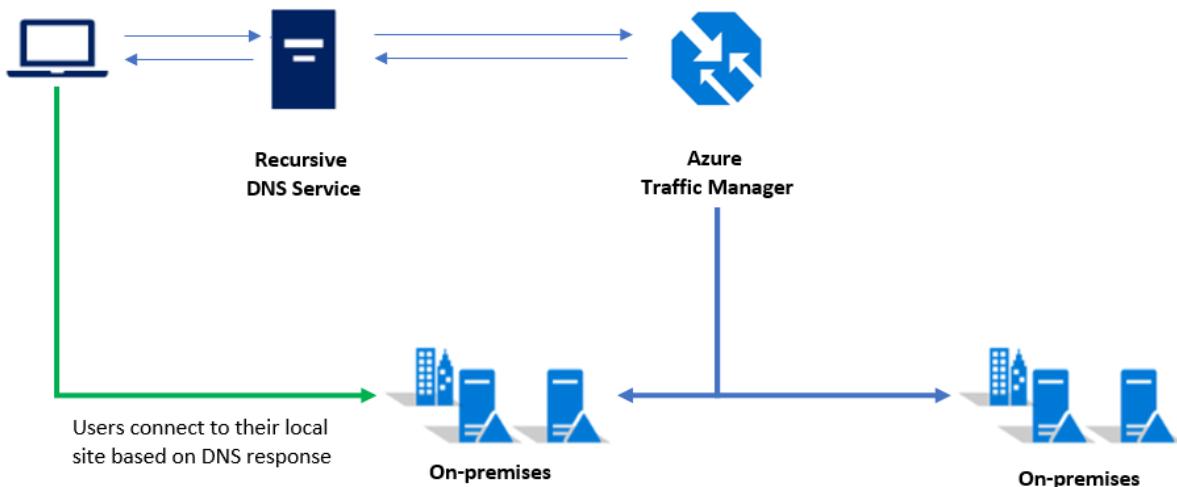
Configure Traffic Manager

To distribute traffic between two (or more) deployments of the application, we'll use [Azure Traffic Manager](#). Azure Traffic Manager is a DNS-based traffic load balancer in Azure.

NOTE

Traffic Manager uses DNS to direct client requests to the most appropriate service endpoint, based on a traffic-routing method and the health of the endpoints.

Instead of using Azure Traffic Manager you can also use other global load-balancing solutions hosted on-premises. In the sample scenario, we'll use Azure Traffic Manager to distribute traffic between two instances of our application. They can run on Azure Stack Hub instances in the same or different locations:



In Azure, we configure Traffic Manager to point to the two different instances of our application:

Name	Status	Monitor status	Type	Weight
akse-seattle	Enabled	Online	External endpoint	1
akse-orlando	Enabled	Checking endpoint	External endpoint	2

As you can see, the two endpoints point to the two instances of the deployed application from the [previous section](#).

At this point:

- The Kubernetes infrastructure has been created, including an ingress controller.
- Clusters have been deployed across two Azure Stack Hub instances.
- Monitoring has been configured.
- Azure Traffic Manager will load balance traffic across the two Azure Stack Hub instances.
- On top of this infrastructure, the sample three-tier application has been deployed in an automated way using Helm Charts.

The solution should now be up and accessible to users!

There are also some post-deployment operational considerations worth discussing, which are covered in the next two sections.

Upgrade Kubernetes

Consider the following topics when upgrading the Kubernetes cluster:

- Upgrading a Kubernetes cluster is a complex Day 2 operation that can be done using AKS Engine. For more information, see [Upgrade a Kubernetes cluster on Azure Stack Hub](#).
- AKS Engine allows you to upgrade clusters to newer Kubernetes and base OS image versions. For more information, see [Steps to upgrade to a newer Kubernetes version](#).
- You can also upgrade only the underlying nodes to newer base OS image versions. For more information, see [Steps to only upgrade the OS image](#).

Newer base OS images contain security and kernel updates. It's the cluster operator's responsibility to monitor the availability of newer Kubernetes Versions and OS Images. The operator should plan and execute these upgrades using AKS Engine. The base OS images must be downloaded from the Azure Stack Hub Marketplace by the Azure Stack Hub Operator.

Scale Kubernetes

Scale is another Day 2 operation that can be orchestrated using AKS Engine.

The scale command reuses your cluster configuration file (`apimodel.json`) in the output directory, as input for a new Azure Resource Manager deployment. AKS Engine executes the scale operation against a specific agent pool. When the scale operation is complete, AKS Engine updates the cluster definition in that same `apimodel.json` file. The cluster definition reflects the new node count in order to reflect the updated, current cluster configuration.

- [Scale a Kubernetes cluster on Azure Stack Hub](#)

Next steps

- Learn more about [Hybrid app design considerations](#).
- Review and propose improvements to [the code for this sample on GitHub](#).

Deploy a highly available MongoDB solution across two Azure Stack Hub environments

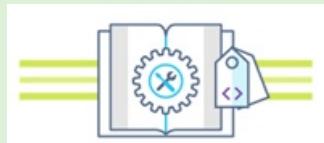
3/10/2022 • 3 minutes to read • [Edit Online](#)

This article will step you through an automated deployment of a basic highly available (HA) MongoDB cluster with a disaster recovery (DR) site across two Azure Stack Hub environments. To learn more about MongoDB and high availability, see [Replica Set Members](#).

In this solution, you'll create a sample environment to:

- Orchestrate a deployment across two Azure Stack Hubs.
- Use Docker to minimize dependency issues with Azure API profiles.
- Deploy a basic highly available MongoDB cluster with a disaster recovery site.

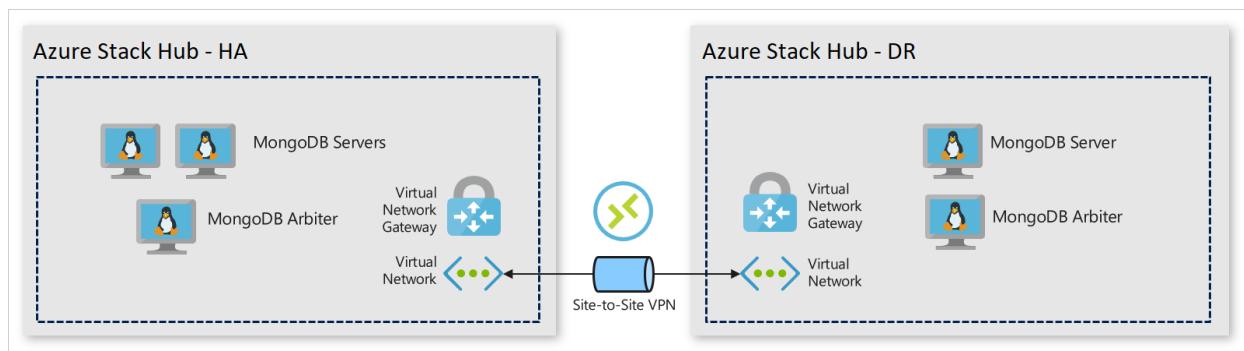
TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that lets you build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Architecture for MongoDB with Azure Stack Hub



Prerequisites for MongoDB with Azure Stack Hub

- Two connected Azure Stack Hub integrated systems (Azure Stack Hub). This deployment doesn't work on the Azure Stack Development Kit (ASDK). To learn more about Azure Stack Hub, see [What is Azure Stack Hub?](#)
 - A tenant subscription on each Azure Stack Hub.
 - **Make a note of each subscription ID and the Azure Resource Manager endpoint for each Azure Stack Hub.**
- An Azure Active Directory (Azure AD) service principal that has permissions to the tenant subscription on each Azure Stack Hub. You may need to create two service principals if the Azure Stack Hubs are deployed against different Azure AD tenants. To learn how to create a service principal for Azure Stack Hub, see [Use an](#)

- app identity to access Azure Stack Hub resources.
- Make a note of each service principal's application ID, client secret, and tenant name (xxxxx.onmicrosoft.com).
- Ubuntu 16.04 syndicated to each Azure Stack Hub's Marketplace. To learn more about marketplace syndication, see [Download Marketplace items to Azure Stack Hub](#).
- [Docker for Windows](#) installed on your local machine.

Get the Docker image

Docker images for each deployment eliminate dependency issues between different versions of Azure PowerShell.

1. Make sure that Docker for Windows is using Windows containers.
2. Run the following command in an elevated command prompt to get the Docker container with the deployment scripts:

```
docker pull intelligentedge/mongodb-hadr:1.0.0
```

Deploy the clusters

1. Once the container image has been successfully pulled, start the image:

```
docker run -it intelligentedge/mongodb-hadr:1.0.0 powershell
```

2. Once the container has started, you'll be given an elevated PowerShell terminal in the container. Change directories to get to the deployment script:

```
cd .\MongoHADRDemo\
```

3. Run the deployment. Provide credentials and resource names where needed. HA refers to the Azure Stack Hub where the HA cluster will be deployed. DR refers to the Azure Stack Hub where the DR cluster will be deployed:

```
.\Deploy-AzureResourceGroup.ps1 `  
-AzureStackApplicationId_HA "applicationIDforHAServicePrincipal" `  
-AzureStackApplicationSercet_HA "clientSecretforHAServicePrincipal" `  
-AADTenantName_HA "hatenantname.onmicrosoft.com" `  
-AzureStackResourceGroup_HA "haresourcegroupname" `  
-AzureStackArmEndpoint_HA "https://management.haazurestack.com" `  
-AzureStackSubscriptionId_HA "haSubscriptionId" `  
-AzureStackApplicationId_DR "applicationIDforDRServicePrincipal" `  
-AzureStackApplicationSercet_DR "ClientSecretforDRServicePrincipal" `  
-AADTenantName_DR "drtenantname.onmicrosoft.com" `  
-AzureStackResourceGroup_DR "drresourcegroupname" `  
-AzureStackArmEndpoint_DR "https://management.drazurestack.com" `  
-AzureStackSubscriptionId_DR "drSubscriptionId"
```

4. Type Y to allow the NuGet provider to be installed, which will kick off the API Profile "2018-03-01-hybrid" modules to be installed.
5. The HA resources will deploy first. Monitor the deployment and wait for it to finish. Once you have the message stating that the HA deployment is finished, you can check the HA Azure Stack Hub's portal to see the resources deployed.

6. Continue with the deployment of DR resources and decide if you'd like to enable a jump box on the DR Azure Stack Hub to interact with the cluster.
7. Wait for DR resource deployment to finish.
8. Once DR resource deployment has finished, exit the container:

```
exit
```

Next steps

- If you enabled the jump box VM on the DR Azure Stack Hub, you can connect via SSH and interact with the MongoDB cluster by installing the mongo CLI. To learn more about interacting with MongoDB, see [The mongo Shell](#).
- To learn more about hybrid cloud apps, see [Hybrid Cloud Solutions..](#)
- Modify the code to this sample on [GitHub](#).

Deploy hybrid app with on-premises data that scales cross-cloud

3/10/2022 • 17 minutes to read • [Edit Online](#)

This solution guide shows you how to deploy a hybrid app that spans both Azure and Azure Stack Hub and uses a single on-premises data source.

By using a hybrid cloud solution, you can combine the compliance benefits of a private cloud with the scalability of the public cloud. Your developers can also take advantage of the Microsoft developer ecosystem and apply their skills to the cloud and on-premises environments.

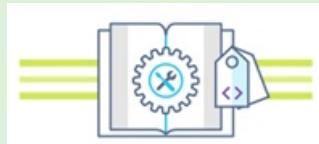
Overview and assumptions

Follow this tutorial to set up a workflow that lets developers deploy an identical web app to a public cloud and a private cloud. This app can access a non-internet routable network hosted on the private cloud. These web apps are monitored and when there's a spike in traffic, a program modifies the DNS records to redirect traffic to the public cloud. When traffic drops to the level before the spike, traffic is routed back to the private cloud.

This tutorial covers the following tasks:

- Deploy a hybrid-connected SQL Server database server.
- Connect a web app in global Azure to a hybrid network.
- Configure DNS for cross-cloud scaling.
- Configure SSL certificates for cross-cloud scaling.
- Configure and deploy the web app.
- Create a Traffic Manager profile and configure it for cross-cloud scaling.
- Set up Application Insights monitoring and alerting for increased traffic.
- Configure automatic traffic switching between global Azure and Azure Stack Hub.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that allows you to build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Assumptions

This tutorial assumes that you have a basic knowledge of global Azure and Azure Stack Hub. If you want to learn more before starting the tutorial, review these articles:

- [Introduction to Azure](#)
- [Azure Stack Hub Key Concepts](#)

This tutorial also assumes that you have an Azure subscription. If you don't have a subscription, [create a free account](#) before you begin.

Prerequisites

Before you start this solution, make sure you meet the following requirements:

- An Azure Stack Development Kit (ASDK) or a subscription on an Azure Stack Hub Integrated System. To deploy the ASDK, follow the instructions in [Deploy the ASDK using the installer](#).
- Your Azure Stack Hub installation should have the following installed:
 - The Azure App Service. Work with your Azure Stack Hub Operator to deploy and configure the Azure App Service on your environment. This tutorial requires the App Service to have at least one (1) available dedicated worker role.
 - A Windows Server 2016 image.
 - A Windows Server 2016 with a Microsoft SQL Server image.
 - The appropriate plans and offers.
 - A domain name for your web app. If you don't have a domain name, you can buy one from a domain provider like GoDaddy, Bluehost, and InMotion.
- An SSL certificate for your domain from a trusted certificate authority like LetsEncrypt.
- A web app that communicates with a SQL Server database and supports Application Insights. You can download the [dotnetcore-sqldb-tutorial](#) sample app from GitHub.
- A hybrid network between an Azure virtual network and Azure Stack Hub virtual network. For detailed instructions, see [Configure hybrid cloud connectivity with Azure and Azure Stack Hub](#).
- A hybrid continuous integration/continuous deployment (CI/CD) pipeline with a private build agent on Azure Stack Hub. For detailed instructions, see [Configure hybrid cloud identity with Azure and Azure Stack Hub apps](#).

Deploy a hybrid-connected SQL Server database server

1. Sign to the Azure Stack Hub user portal.
2. On the **Dashboard**, select **Marketplace**.

Microsoft Azure Stack

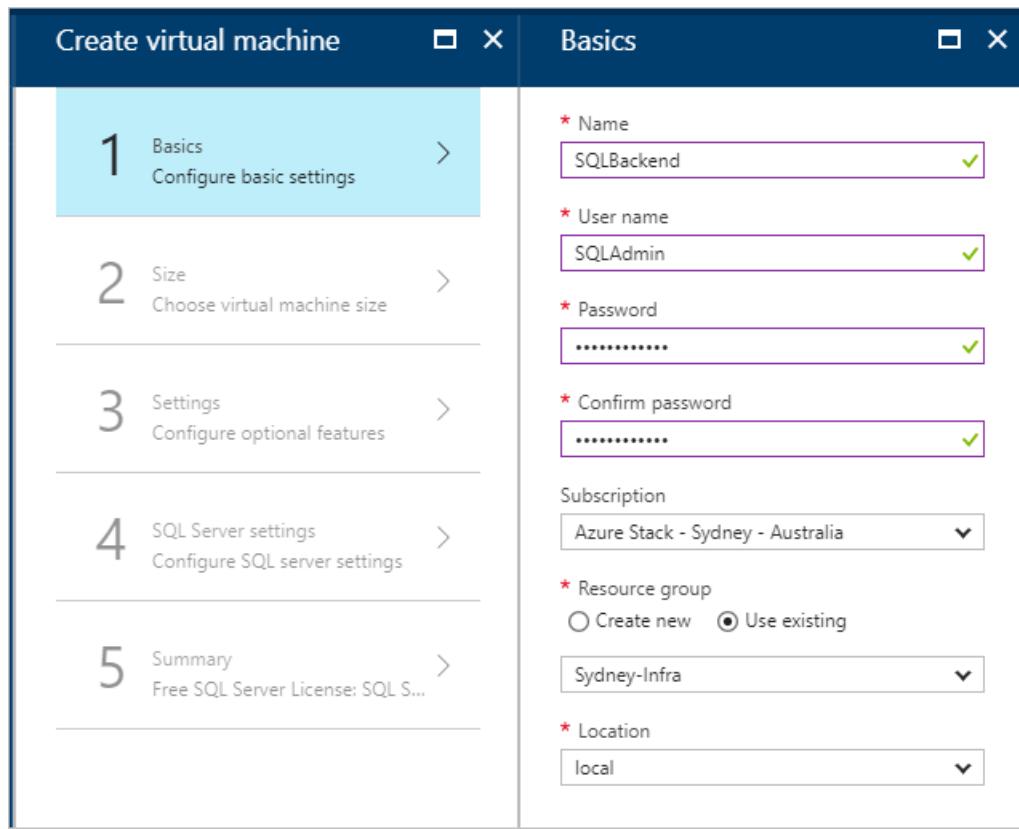
The screenshot shows the Microsoft Azure Stack dashboard. On the left sidebar, there are several navigation items: New, Dashboard, All resources, Resource groups, and App Services. The main area is titled "Dashboard" and features a "Marketplace" section. Within the Marketplace section, there is a blue shopping bag icon with a white cloud inside it, and the word "Marketplace" below it.

3. In Marketplace, select Compute, and then choose More. Under More, select the Free SQL Server License: SQL Server 2017 Developer on Windows Server image.

The screenshot shows the Microsoft Azure Stack Marketplace Compute page. The left sidebar has a "Compute" category selected. The main area displays a list of compute-related offerings. One item is highlighted: "Free SQL Server License: SQL Server 2017 Developer on Windows Server". This item is shown with a red icon and a brief description: "Free SQL Server License: SQL Server 2017 Developer on Windows Server Microsoft". To the right of this item, there is a vertical list of other compute offerings, each with its own icon and name.

Offer	Description	Provider
SQL Server 2017 Eval on Ubuntu Server 16.04 LTS ASDK	Eval on Ubuntu Server 16.04 LTS ASDK	Microsoft
MySQL 5.7 on Ubuntu Server ASDK	MySQL 5.7 on Ubuntu Server ASDK	MySQL
Canonical-UbuntuServer-16.04-LTS	Canonical-UbuntuServer-16.04-LTS	Canonical
Availability Set	Availability Set	Microsoft
Free SQL Server License: SQL Server 2017 Developer on SUSE Linux	Free SQL Server License: SQL Server 2017 Developer on SUSE Linux	Microsoft
Free SQL Server License: SQL Server 2017 Developer on Ubuntu Server	Free SQL Server License: SQL Server 2017 Developer on Ubuntu Server	Microsoft
Free SQL Server License: SQL Server 2017 Developer on Windows Server	Free SQL Server License: SQL Server 2017 Developer on Windows Server	Microsoft
Windows Server 2016 Datacenter	Windows Server 2016 Datacenter	Microsoft
Windows Server 2016 Datacenter Core Eval	Windows Server 2016 Datacenter Core Eval	MicrosoftWindowsServer
Windows Server 2016 Datacenter Eval	Windows Server 2016 Datacenter Eval	MicrosoftWindowsServer
CentOS-based 7.4	CentOS-based 7.4	Rogue Wave Software (formerly OpenLogic)
SLES 12 SP3 (BIOS)	SLES 12 SP3 (BIOS)	SUSE

4. On Free SQL Server License: SQL Server 2017 Developer on Windows Server, select Create.
5. On Basics > Configure basic settings, provide a Name for the virtual machine (VM), a User name for the SQL Server SA, and a Password for the SA. From the Subscription drop-down list, select the subscription that you're deploying to. For Resource group, use Choose existing and put the VM in the same resource group as your Azure Stack Hub web app.



6. Under **Size**, pick a size for your VM. For this tutorial, we recommend A2_Standard or a DS2_V2_Standard.

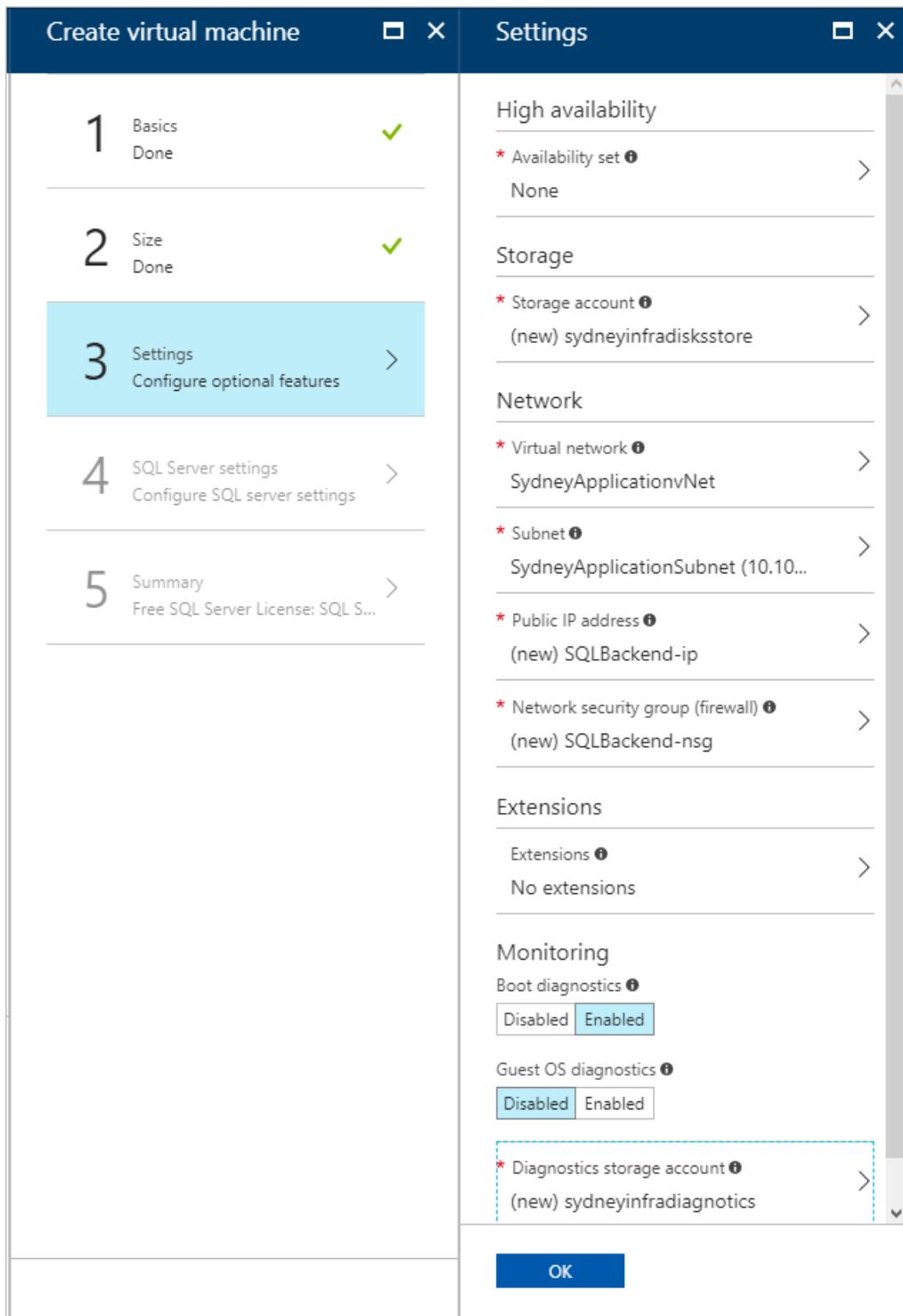
7. Under **Settings > Configure optional features**, configure the following settings:

- **Storage account:** Create a new account if you need one.
- **Virtual network:**

IMPORTANT

Make sure your SQL Server VM is deployed on the same virtual network as the VPN gateways.

- **Public IP address:** Use the default settings.
- **Network security group:** (NSG). Create a new NSG.
- **Extensions and Monitoring:** Keep the default settings.
- **Diagnostics storage account:** Create a new account if you need one.
- Select **OK** to save your configuration.



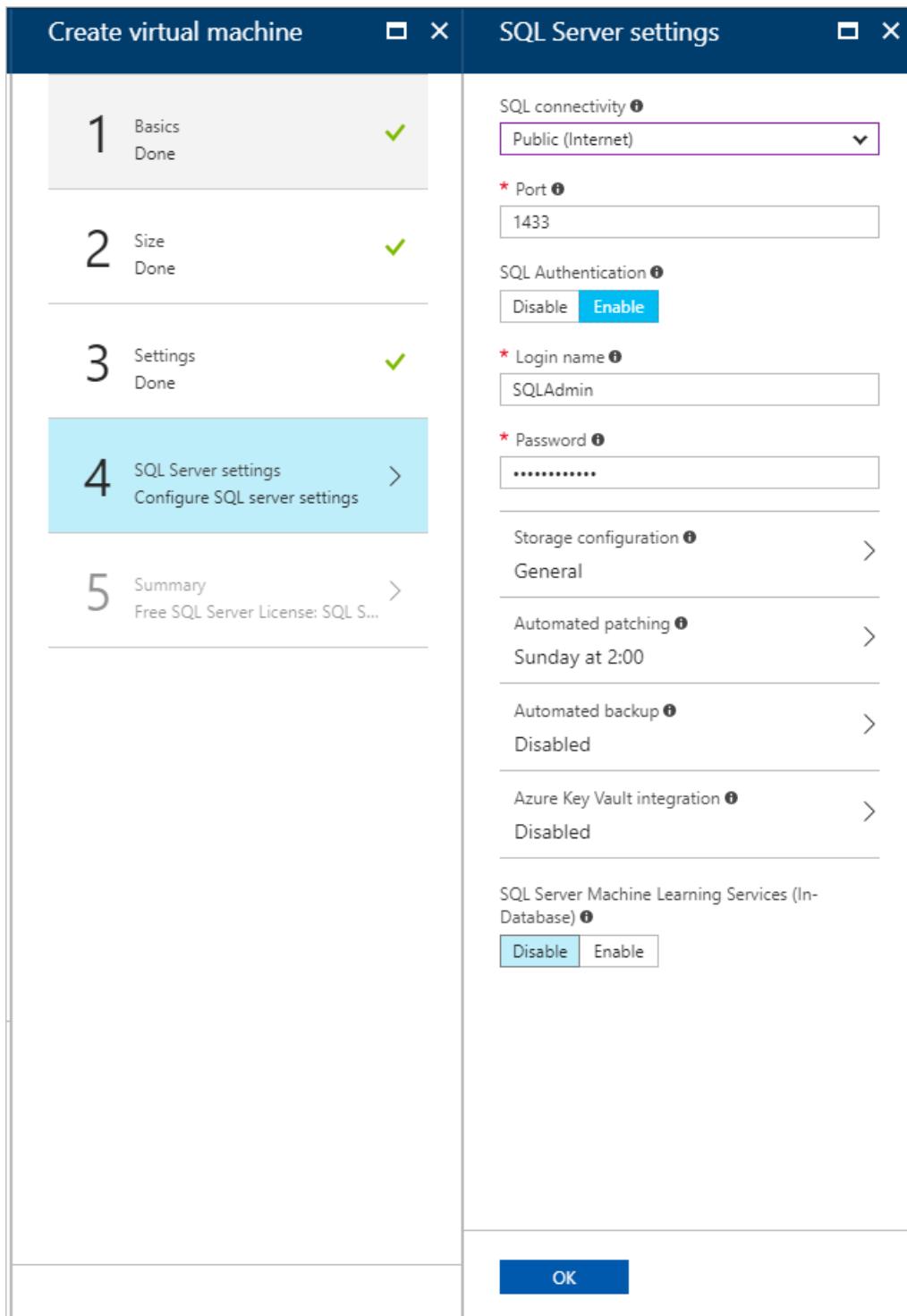
8. Under **SQL Server settings**, configure the following settings:

- For **SQL connectivity**, select **Public (Internet)**.
- For **Port**, keep the default, **1433**.
- For **SQL authentication**, select **Enable**.

NOTE

When you enable SQL authentication, it should auto-populate with the "SQLAdmin" information that you configured in **Basics**.

- For the rest of the settings, keep the defaults. Select **OK**.



9. On **Summary**, review the VM configuration and then select **OK** to start the deployment.

Summary

Validation passed

Basics

Subscription	Azure Stack - Sydney - Australia
Resource group	Sydney-Infra
Location	local

Settings

Computer name	SQLBackend
Disk type	HDD
User name	SQLAdmin
Size	Standard A2
Storage account	(new) sydneyinfradisksstore
Managed	No
Virtual network	SydneyApplicationvNet
Subnet	SydneyApplicationSubnet (10.103.100.0/24)
Public IP address	(new) SQLBackend-ip
Network security group (firewall)	(new) SQLBackend-nsg
Availability set	None
Guest OS diagnostics	Disabled
Boot diagnostics	Enabled
Diagnostics storage account	(new) sydneyinfradiagnostics

SQL Server settings

SQL connectivity level	Public
SQL port	1433
SQL Authentication	Enabled
SQL Authentication login	SQLAdmin
SQL Server Machine Learning S...	Disabled
Storage optimization type	General
Storage size	128 (GB)
Automated patching	Enabled
Auto patching schedule	Sunday at 2:00
Automated backup	Disabled
Azure Key Vault integration	Disabled

OK [Download template and parameters](#)

10. It takes some time to create the new VM. You can view the STATUS of your VMs in **Virtual machines**.

Microsoft Azure Stack [Virtual machines](#) [Search resources](#)

Virtual machines
Northwind Cloud

New [Add](#) [Columns](#) [Refresh](#)

Subscriptions: All 6 selected
[Filter by name...](#) [All subscriptions](#) [All resource groups](#)

NAME	STATUS	RESOURCE GROUP	LOCATION
BuildServer01	Running	BuildServerInfra	local
BuildServer02	Running	BuildServerInfra	local
SQLBackend	Creating	Sydney-Infra	local

Create web apps in Azure and Azure Stack Hub

The Azure App Service simplifies running and managing a web app. Because Azure Stack Hub is consistent with Azure, the App Service can run in both environments. You'll use the App Service to host your app.

Create web apps

1. Create a web app in Azure by following the instructions in [Manage an App Service plan in Azure](#). Make sure you put the web app in the same subscription and resource group as your hybrid network.
2. Repeat the previous step (1) in Azure Stack Hub.

Add route for Azure Stack Hub

The App Service on Azure Stack Hub must be routable from the public internet to let users access your app. If your Azure Stack Hub is accessible from the internet, make a note of the public-facing IP address or URL for the Azure Stack Hub web app.

If you're using an ASDK, you can [configure a static NAT mapping](#) to expose App Service outside the virtual environment.

Connect a web app in Azure to a hybrid network

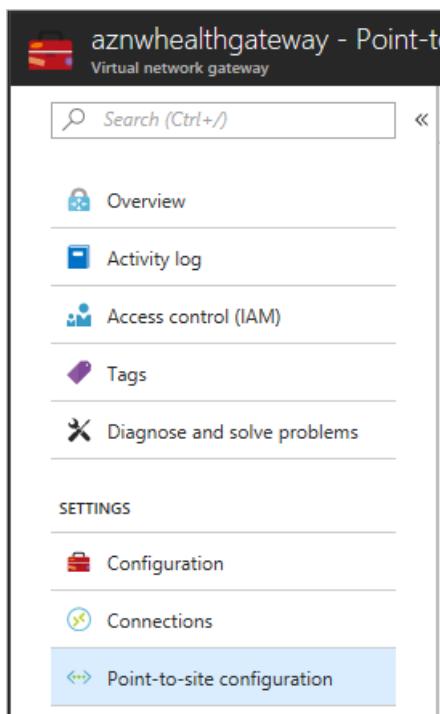
To provide connectivity between the web front end in Azure and the SQL Server database in Azure Stack Hub, the web app must be connected to the hybrid network between Azure and Azure Stack Hub. To enable connectivity, you'll have to:

- Configure point-to-site connectivity.
- Configure the web app.
- Modify the local network gateway in Azure Stack Hub.

Configure the Azure virtual network for point-to-site connectivity

The virtual network gateway in the Azure side of the hybrid network must allow point-to-site connections to integrate with Azure App Service.

1. In the Azure portal, go to the virtual network gateway page. Under **Settings**, select **Point-to-site configuration**.



2. Select **Configure now** to configure point-to-site.

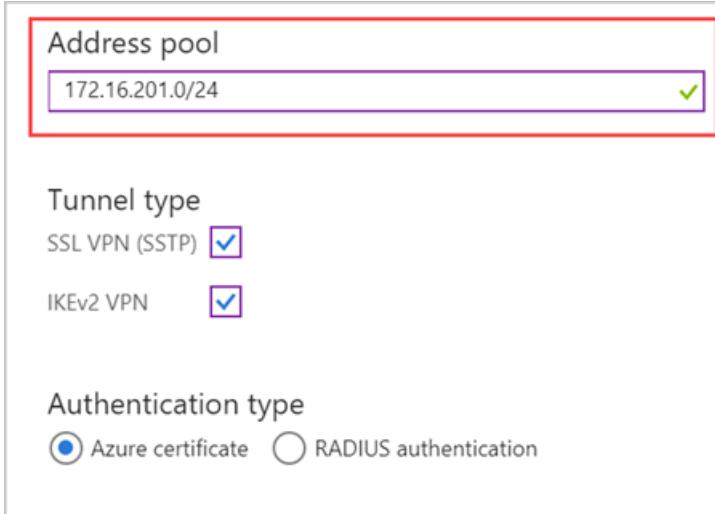


3. On the **Point-to-site** configuration page, enter the private IP address range that you want to use in **Address pool**.

NOTE

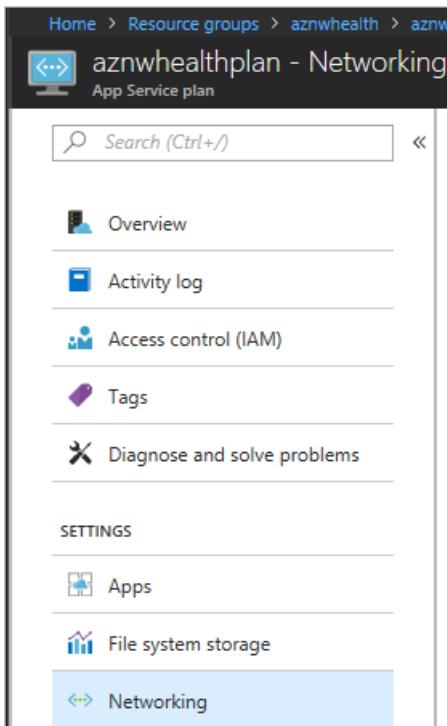
Make sure that the range you specify doesn't overlap with any of the address ranges already used by subnets in the global Azure or Azure Stack Hub components of the hybrid network.

Under **Tunnel Type**, uncheck the **IKEv2 VPN**. Select **Save** to finish configuring point-to-site.



Integrate the Azure App Service app with the hybrid network

1. To connect the app to the Azure VNet, follow the instructions in [Gateway required VNet integration](#).
2. Go to **Settings** for the App Service plan hosting the web app. In **Settings**, select **Networking**.



3. In **VNET Integration**, select [Click here to manage](#).



VNET Integration
1 VNET configured.

Securely access resources available in or through your Azure VNET

[Click here to manage](#)

- Select the VNET that you want to configure. Under **IP ADDRESSES ROUTED TO VNET**, enter the IP address range for the Azure VNet, the Azure Stack Hub VNet, and the point-to-site address spaces. Select **Save** to validate and save these settings.

Virtual Network Integration
aznwhealthplan

App Service Plan	aznwhealthplan
App Service Plan Location	West US
VNETs integrated with	1 out of 5
VNET NAME	GATEWAY STATUS
aznwhealtnetwork	Online

Virtual Network Integrati...
aznwhealthplan

Save **Discard** **Sync Network**

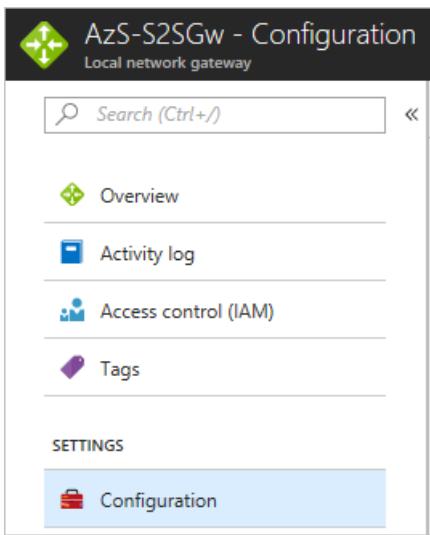
VNET NAME	aznwhealtnetwork
LOCATION	West US
CERTIFICATE STATUS	Certificates in sync
GATEWAY STATUS	Online
APPS USING VIRTUAL NETWORK	
aznwhealth	
VNET ADDRESS SPACE	
START ADDRESS	END ADDRESS
10.0.102.0	10.0.103.255
POINT-TO-SITE ADDRESS SPACE	
START ADDRESS	END ADDRESS
10.0.0.0	10.0.0.255
SITE-TO-SITE ADDRESS SPACE	
START ADDRESS	END ADDRESS
10.0.100.0	10.0.101.255
DNS SERVERS	
No DNS servers are configured on this VNET.	
IP ADDRESSES ROUTED TO VNET	
START ADDRESS	END ADDRESS
10.0.102.0	10.0.103.255
10.0.0.0	10.0.0.255
10.0.100.0	10.0.101.255
...	

To learn more about how App Service integrates with Azure VNets, see [Integrate your app with an Azure Virtual Network](#).

Configure the Azure Stack Hub virtual network

The local network gateway in the Azure Stack Hub virtual network needs to be configured to route traffic from the App Service point-to-site address range.

- In the Azure Stack Hub portal, go to **Local network gateway**. Under **Settings**, select **Configuration**.



2. In **Address space**, enter the point-to-site address range for the virtual network gateway in Azure.

* IP address ⓘ
161.47.116.91

Address space ⓘ

10.0.100.0/23

10.0.0.0/24

Add additional address range

Configure BGP settings

3. Select **Save** to validate and save the configuration.

Configure DNS for cross-cloud scaling

By properly configuring DNS for cross-cloud apps, users can access the global Azure and Azure Stack Hub instances of your web app. The DNS configuration for this tutorial also lets Azure Traffic Manager route traffic when the load increases or decreases.

This tutorial uses Azure DNS to manage the DNS because App Service domains won't work.

Create subdomains

Because Traffic Manager relies on DNS CNAMEs, a subdomain is needed to properly route traffic to endpoints. For more information about DNS records and domain mapping, see [map domains with Traffic Manager](#).

For the Azure endpoint, you'll create a subdomain that users can use to access your web app. For this tutorial, can use `app.northwind.com`, but you should customize this value based on your own domain.

You'll also need to create a subdomain with an A record for the Azure Stack Hub endpoint. You can use `azurestack.northwind.com`.

Configure a custom domain in Azure

1. Add the `app.northwind.com` hostname to the Azure web app by [mapping a CNAME to Azure App Service](#).

Configure custom domains in Azure Stack Hub

1. Add the `azurestack.northwind.com` hostname to the Azure Stack Hub web app by [mapping an A record to Azure App Service](#). Use the internet-routable IP address for the App Service app.

2. Add the **app.northwind.com** hostname to the Azure Stack Hub web app by [mapping a CNAME to Azure App Service](#). Use the hostname you configured in the previous step (1) as the target for the CNAME.

Configure SSL certificates for cross-cloud scaling

It's important to ensure sensitive data collected by your web app is secure in transit to and when stored on the SQL database.

You'll configure your Azure and Azure Stack Hub web apps to use SSL certificates for all incoming traffic.

Add SSL to Azure and Azure Stack Hub

To add SSL to Azure:

1. Make sure that the SSL certificate you get is valid for the subdomain you created. (It's okay to use wildcard certificates.)
2. In the Azure portal, follow the instructions in the **Prepare your web app and Bind your SSL certificate** sections of the [Bind an existing custom SSL certificate to Azure Web Apps](#) article. Select **SNI-based SSL** as the **SSL Type**.
3. Redirect all traffic to the HTTPS port. Follow the instructions in the **Enforce HTTPS** section of the [Bind an existing custom SSL certificate to Azure Web Apps](#) article.

To add SSL to Azure Stack Hub:

1. Repeat steps 1-3 that you used for Azure, using the Azure Stack Hub portal.

Configure and deploy the web app

You'll configure the app code to report telemetry to the correct Application Insights instance and configure the web apps with the right connection strings. To learn more about Application Insights, see [What is Application Insights?](#)

Add Application Insights

1. Open your web app in Microsoft Visual Studio.
2. [Add Application Insights](#) to your project to transmit the telemetry that Application Insights uses to create alerts when web traffic increases or decreases.

Configure dynamic connection strings

Each instance of the web app will use a different method to connect to the SQL database. The app in Azure uses the private IP address of the SQL Server VM and the app in Azure Stack Hub uses the public IP address of the SQL Server VM.

NOTE

On an Azure Stack Hub integrated system, the public IP address shouldn't be internet-routable. On an ASDK, the public IP address isn't routable outside the ASDK.

You can use App Service environment variables to pass a different connection string to each instance of the app.

1. Open the app in Visual Studio.
2. Open Startup.cs and find the following code block:

```
services.AddDbContext<MyDatabaseContext>(options =>
    options.UseSqlite("Data Source=localdatabase.db"));
```

- Replace the previous code block with the following code, which uses a connection string defined in the `appsettings.json` file:

```
services.AddDbContext<MyDatabaseContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection")));
// Automatically perform database migration
services.BuildServiceProvider().GetService<MyDatabaseContext>().Database.Migrate();
```

Configure App Service app settings

- Create connection strings for Azure and Azure Stack Hub. The strings should be the same, except for the IP addresses that are used.
- In Azure and Azure Stack Hub, add the appropriate connection string [as an app setting](#) in the web app, using `SQLCONNSTR_` as a prefix in the name.
- Save** the web app settings and restart the app.

Enable automatic scaling in global Azure

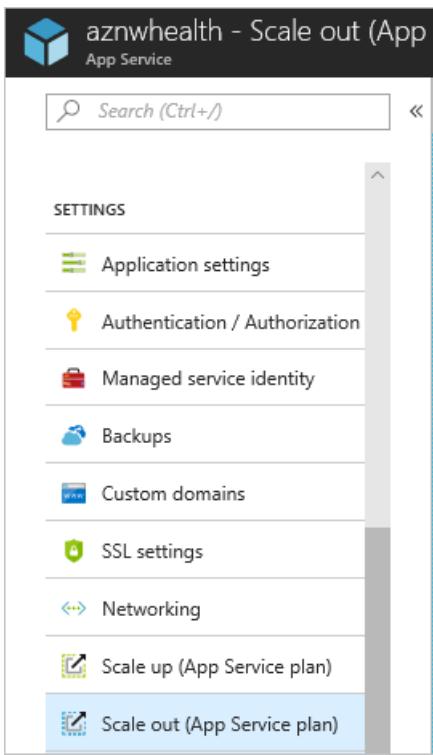
When you create your web app in an App Service environment, it starts with one instance. You can automatically scale out to add instances to provide more compute resources for your app. Similarly, you can automatically scale in and reduce the number of instances your app needs.

NOTE

You need to have an App Service plan to configure scale out and scale in. If you don't have a plan, create one before starting the next steps.

Enable automatic scale-out

- In the Azure portal, find the App Service plan for the sites you want to scale out, and then select **Scale-out (App Service plan)**.



2. Select **Enable autoscale**.

A screenshot of the 'Autoscale' configuration page. At the top, there are buttons for Save, Discard, Disable autoscale, and Refresh. Below that, tabs for Configure, Run history, JSON, and Notify are visible, with 'Configure' being the active tab. The main area contains sections for 'Override condition' and 'Instance count'. A message at the bottom states: 'Your autoscale configuration is disabled. To reinstate your configuration, enable autoscale.' A prominent blue button labeled 'Enable autoscale' is centered at the bottom of the page.

3. Enter a name for **Autoscale Setting Name**. For the **Default** auto scale rule, select **Scale based on a metric**. Set the **Instance limits** to **Minimum: 1**, **Maximum: 10**, and **Default: 1**.

Autoscale setting name: enrollment-autoscale

Resource group: aznwhealth

Instance count: 1

Default Auto created scale condition (edit) (delete)

Delete warning: (i) The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.

Scale mode: Scale based on a metric Scale to a specific instance count

Rules

When	aznwhealthplan	(Average) CpuPercentage > 50	Increase instance count by 2
Scale in			
When	aznwhealthplan	(Average) CpuPercentage < 30	Decrease instance count by 1

[+ Add a rule](#)

Instance limits

Minimum (i)	1	Maximum (i)	10	Default (i)	1
--	---	--	----	--	---

Schedule

This scale condition is executed when none of the other scale condition(s) match

4. Select **+Add a rule**.

5. In **Metric Source**, select **Current Resource**. Use the following Criteria and Actions for the rule.

Criteria

1. Under **Time Aggregation**, select **Average**.
2. Under **Metric Name**, select **CPU Percentage**.
3. Under **Operator**, select **Greater than**.
 - Set the **Threshold** to **50**.
 - Set the **Duration** to **10**.

Action

1. Under **Operation**, select **Increase Count by**.
2. Set the **Instance Count** to **2**.
3. Set the **Cool down** to **5**.
4. Select **Add**.
5. Select the **+ Add a rule**.
6. In **Metric Source**, select **Current Resource**.

NOTE

The current resource will contain your App Service plan's name/GUID and the **Resource Type** and **Resource** drop-down lists will be unavailable.

Enable automatic scale in

When traffic decreases, the Azure web app can automatically reduce the number of active instances to reduce costs. This action is less aggressive than scale-out and minimizes the impact on app users.

1. Go to the **Default** scale out condition, then select **+ Add a rule**. Use the following Criteria and Actions for the rule.

Criteria

1. Under **Time Aggregation**, select **Average**.
2. Under **Metric Name**, select **CPU Percentage**.
3. Under **Operator**, select **Less than**.
 - Set the **Threshold** to 30.
 - Set the **Duration** to 10.

Action

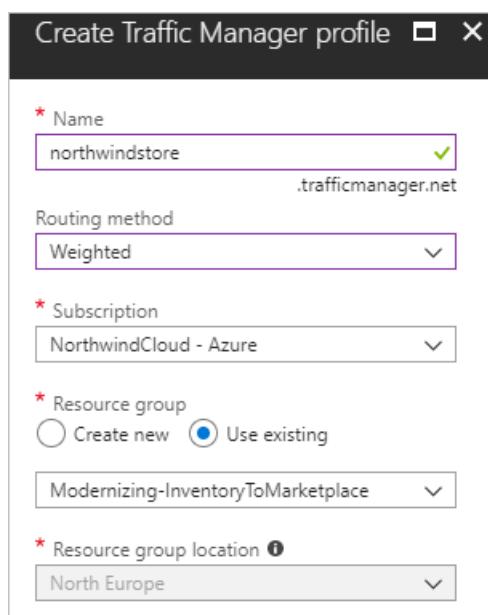
1. Under **Operation**, select **Decrease Count by**.
 - Set the **Instance Count** to 1.
 - Set the **Cool down** to 5.
2. Select **Add**.

Create a Traffic Manager profile and configure cross-cloud scaling

Create a Traffic Manager profile using the Azure portal, then configure endpoints to enable cross-cloud scaling.

Create Traffic Manager profile

1. Select **Create a resource**.
2. Select **Networking**.
3. Select **Traffic Manager profile** and configure the following settings:
 - In **Name**, enter a name for your profile. This name **must** be unique in the trafficmanager.net zone and is used to create a new DNS name (for example, northwindstore.trafficmanager.net).
 - For **Routing method**, select the **Weighted**.
 - For **Subscription**, select the subscription you want to create this profile in.
 - In **Resource Group**, create a new resource group for this profile.
 - In **Resource group location**, select the location of the resource group. This setting refers to the location of the resource group and has no impact on the Traffic Manager profile that's deployed globally.
4. Select **Create**.



When the global deployment of your Traffic Manager profile is complete, it's shown in the list of

resources for the resource group you created it under.

Add Traffic Manager endpoints

1. Search for the Traffic Manager profile you created. If you navigated to the resource group for the profile, select the profile.
2. In **Traffic Manager profile**, under **SETTINGS**, select **Endpoints**.
3. Select **Add**.
4. In **Add endpoint**, use the following settings for Azure Stack Hub:
 - For **Type**, select **External endpoint**.
 - Enter a **Name** for the endpoint.
 - For **Fully qualified domain name (FQDN) or IP**, enter the external URL for your Azure Stack Hub web app.
 - For **Weight**, keep the default, 1. This weight results in all traffic going to this endpoint if it's healthy.
 - Leave **Add as disabled** unchecked.
5. Select **OK** to save the Azure Stack Hub endpoint.

You'll configure the Azure endpoint next.

1. On **Traffic Manager profile**, select **Endpoints**.
2. Select **+Add**.
3. On **Add endpoint**, use the following settings for Azure:
 - For **Type**, select **Azure endpoint**.
 - Enter a **Name** for the endpoint.
 - For **Target resource type**, select **App Service**.
 - For **Target resource**, select **Choose an app service** to see a list of Web Apps in the same subscription.
 - In **Resource**, pick the App service that you want to add as the first endpoint.
 - For **Weight**, select 2. This setting results in all traffic going to this endpoint if the primary endpoint is unhealthy, or if you have a rule/alert that redirects traffic when triggered.
 - Leave **Add as disabled** unchecked.
4. Select **OK** to save the Azure endpoint.

After both endpoints are configured, they're listed in **Traffic Manager profile** when you select **Endpoints**. The example in the following screen capture shows two endpoints, with status and configuration information for each one.

NAME	STATUS	MONITO...	TYPE	WEIG...
NorthwindCloudShop-AzureStack	Enabled	Degraded	External endpoint	1
NorthwindCloudShop-Azure	Enabled	Online	Azure endpoint	2

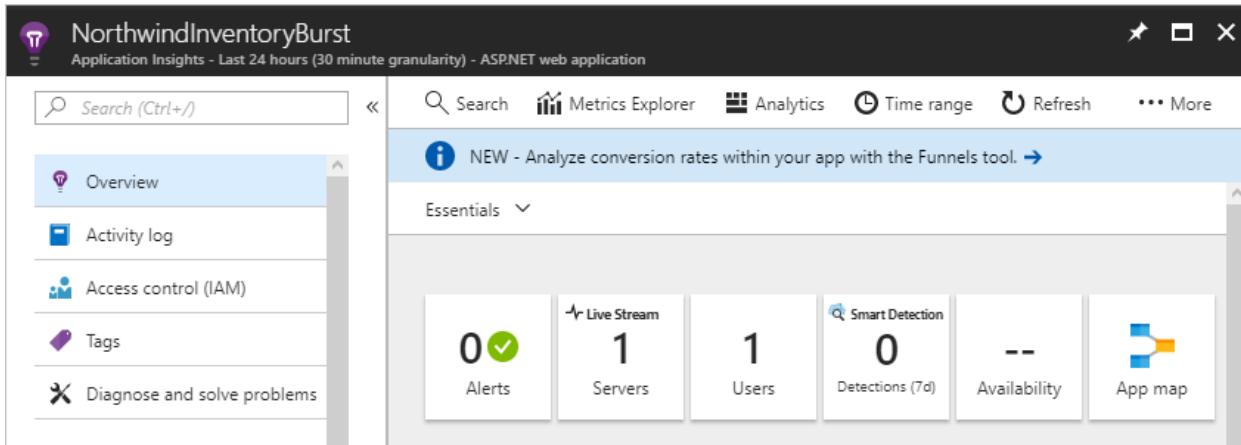
Set up Application Insights monitoring and alerting in Azure

Azure Application Insights lets you monitor your app and send alerts based on conditions you configure. Some examples are: the app is unavailable, is experiencing failures, or is showing performance issues.

You'll use Azure Application Insights metrics to create alerts. When these alerts trigger, your web app's instance will automatically switch from Azure Stack Hub to Azure to scale out, and then back to Azure Stack Hub to scale in.

Create an alert from metrics

In the Azure portal, go to the resource group for this tutorial, and select the Application Insights instance to open Application Insights.



You'll use this view to create a scale-out alert and a scale-in alert.

Create the scale-out alert

1. Under **CONFIGURE**, select **Alerts (classic)**.
2. Select **Add metric alert (classic)**.
3. In **Add rule**, configure the following settings:
 - For **Name**, enter **Burst into Azure Cloud**.
 - A **Description** is optional.
 - Under **Source > Alert on**, select **Metrics**.
 - Under **Criteria**, select your subscription, the resource group for your Traffic Manager profile, and the name of the Traffic Manager profile for the resource.
4. For **Metric**, select **Request Rate**.
5. For **Condition**, select **Greater than**.
6. For **Threshold**, enter **2**.
7. For **Period**, select **Over the last 5 minutes**.
8. Under **Notify via**:
 - Check the checkbox for **Email owners, contributors, and readers**.
 - Enter your email address for **Additional administrator email(s)**.
9. On the menu bar, select **Save**.

Create the scale-in alert

1. Under **CONFIGURE**, select **Alerts (classic)**.
2. Select **Add metric alert (classic)**.
3. In **Add rule**, configure the following settings:
 - For **Name**, enter **Scale back into Azure Stack Hub**.

- A **Description** is optional.
 - Under **Source > Alert on**, select **Metrics**.
 - Under **Criteria**, select your subscription, the resource group for your Traffic Manager profile, and the name of the Traffic Manager profile for the resource.
4. For **Metric**, select **Request Rate**.
 5. For **Condition**, select **Less than**.
 6. For **Threshold**, enter **2**.
 7. For **Period**, select **Over the last 5 minutes**.
 8. Under **Notify via**:
 - Check the checkbox for **Email owners, contributors, and readers**.
 - Enter your email address for **Additional administrator email(s)**.
 9. On the menu bar, select **Save**.

The following screenshot shows the alerts for scale-out and scale-in.

The screenshot shows the Azure Application Insights Alerts (classic) interface. At the top, there are filters for Subscription (NorthwindCloud - Azure), Source (All sources), Resource group (Modernizing-InventoryToMarketplace), and Resource type (Application Insights). Below the filters, a breadcrumb navigation shows: Home > Application Insights > NorthwindInventoryBurst > Alerts (classic). A message indicates new functionalities are available in the Alerts section. The main table lists two alerts:

NAME	STATUS	CONDITION	RESOURCE GROUP	RESOURCE
Burst into Azure Cloud	Active	Request rate > 2 CountPerSecond	Modernizing-InventoryToMarketplace	NorthwindInventoryBurst
Scale Back to Azure Stack	Active	Request rate < 2 CountPerSecond	Modernizing-InventoryToMarketplace	NorthwindInventoryBurst

Redirect traffic between Azure and Azure Stack Hub

You can configure manual or automatic switching of your web app traffic between Azure and Azure Stack Hub.

Configure manual switching between Azure and Azure Stack Hub

When your web site reaches the thresholds that you configure, you'll receive an alert. Use the following steps to manually redirect traffic to Azure.

1. In the Azure portal, select your Traffic Manager profile.

The screenshot shows the Azure Traffic Manager Endpoints page. At the top, there are buttons for **Add** and **Refresh**. Below the buttons is a search bar labeled "Search endpoints". The main table lists two endpoints:

NAME	STATUS	MONITO...	TYPE	WEIG...
NorthwindCloudShop-AzureStack	Enabled	Degraded	External endpoint	1
NorthwindCloudShop-Azure	Enabled	Online	Azure endpoint	2

2. Select **Endpoints**.
3. Select the **Azure endpoint**.
4. Under **Status**, select **Enabled**, and then select **Save**.

NorthwindCloudShop-Azure
northwindcloudshop

Status
 Disabled Enabled

Monitor status
Online

Type
Azure endpoint

Target resource type

* Target resource
northwindinventory-cloud >

* Weight

5. On Endpoints for the Traffic Manager profile, select **External endpoint**.

6. Under Status, select **Disabled**, and then select **Save**.

NorthwindCloudShop-AzureStack
northwindcloudshop

Status
 Disabled Enabled

Monitor status
Disabled

Type
External endpoint

* Target

* Weight

After the endpoints are configured, app traffic goes to your Azure scale-out web app instead of the Azure Stack Hub web app.

Search endpoints					
NAME	STATUS	MONITOR STATUS	TYPE	WEIGHT	
NorthwindCloudShop-AzureStack	Disabled	Disabled	External endpoint	1	
NorthwindCloudShop-Azure	Enabled	Online	Azure endpoint	2	

To reverse the flow back to Azure Stack Hub, use the previous steps to:

- Enable the Azure Stack Hub endpoint.
- Disable the Azure endpoint.

Configure automatic switching between Azure and Azure Stack Hub

You can also use Application Insights monitoring if your app runs in a [serverless](#) environment provided by Azure Functions.

In this scenario, you can configure Application Insights to use a webhook that calls a function app. This app automatically enables or disables an endpoint in response to an alert.

Use the following steps as a guide to configure automatic traffic switching.

1. Create an Azure Function app.
2. Create an HTTP-triggered function.
3. Import the Azure SDKs for Resource Manager, Web Apps, and Traffic Manager.
4. Develop code to:
 - Authenticate to your Azure subscription.
 - Use a parameter that toggles the Traffic Manager endpoints to direct traffic to Azure or Azure Stack Hub.
5. Save your code and add the function app's URL with the appropriate parameters to the **Webhook** section of the Application Insights alert rule settings.
6. Traffic is automatically redirected when an Application Insights alert fires.

Next steps

- To learn more about Azure Cloud Patterns, see [Cloud Design Patterns](#).

Deploy a SQL Server 2016 availability group across two Azure Stack Hub environments

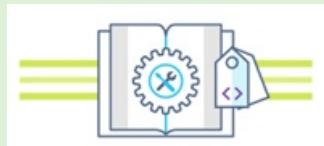
3/10/2022 • 3 minutes to read • [Edit Online](#)

This article will step you through an automated deployment of a basic highly available (HA) SQL Server 2016 Enterprise cluster with an asynchronous disaster recovery (DR) site across two Azure Stack Hub environments. To learn more about SQL Server 2016 and high availability, see [Always On availability groups: a high-availability and disaster-recovery solution](#).

In this solution, you'll build a sample environment to:

- Orchestrate a deployment across two Azure Stack Hubs.
- Use Docker to minimize dependency issues with Azure API profiles.
- Deploy a basic highly available SQL Server 2016 Enterprise cluster with a disaster recovery site.

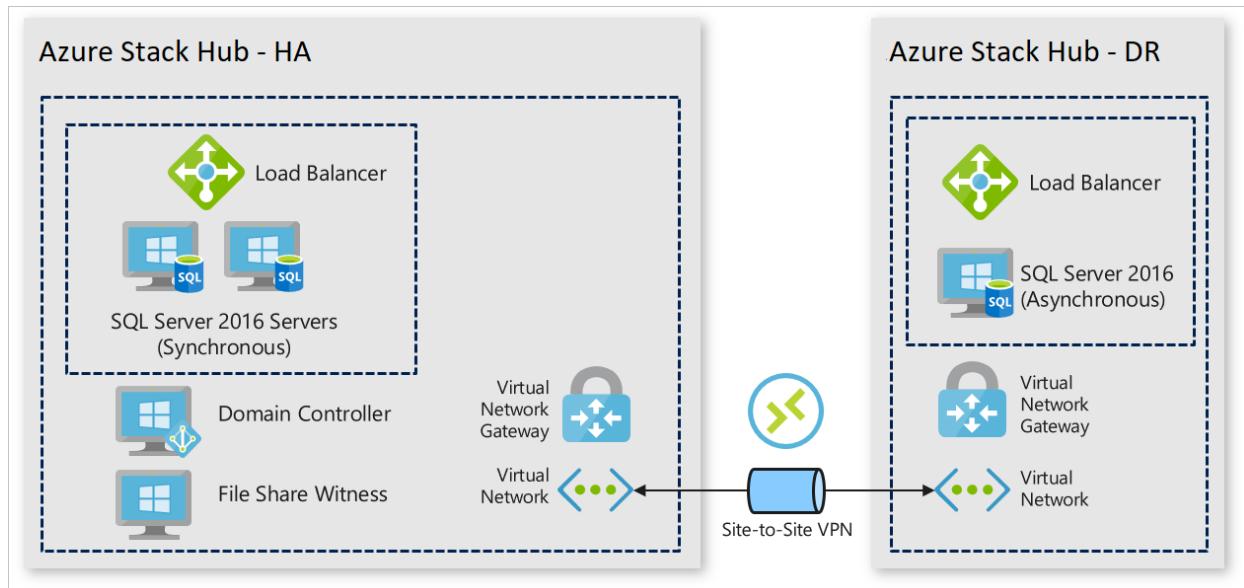
TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that lets you build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Architecture for SQL Server 2016



Prerequisites for SQL Server 2016

- Two connected Azure Stack Hub integrated systems (Azure Stack Hub). This deployment doesn't work on the

Azure Stack Development Kit (ASDK). To learn more about Azure Stack Hub, see the [Azure Stack overview](#).

- A tenant subscription on each Azure Stack Hub.
 - **Make a note of each subscription ID and the Azure Resource Manager endpoint for each Azure Stack Hub.**
- An Azure Active Directory (Azure AD) service principal that has permissions to the tenant subscription on each Azure Stack Hub. You may need to create two service principals if the Azure Stack Hubs are deployed against different Azure AD tenants. To learn how to create a service principal for Azure Stack Hub, see [Create service principals to give apps access to Azure Stack Hub resources](#).
 - **Make a note of each service principal's application ID, client secret, and tenant name (xxxxx.onmicrosoft.com).**
- SQL Server 2016 Enterprise syndicated to each Azure Stack Hub's Marketplace. To learn more about marketplace syndication, see [Download Marketplace items to Azure Stack Hub](#). **Make sure that your organization has the appropriate SQL licenses.**
- [Docker for Windows](#) installed on your local machine.

Get the Docker image

Docker images for each deployment eliminate dependency issues between different versions of Azure PowerShell.

1. Make sure that Docker for Windows is using Windows containers.
2. Run the following script in an elevated command prompt to get the Docker container with the deployment scripts.

```
docker pull intelligentedge/sqlserver2016-hadr:1.0.0
```

Deploy the availability group

1. Once the container image has been successfully pulled, start the image.

```
docker run -it intelligentedge/sqlserver2016-hadr:1.0.0 powershell
```

2. Once the container has started, you'll be given an elevated PowerShell terminal in the container. Change directories to get to the deployment script.

```
cd .\SQLHADRDemo\
```

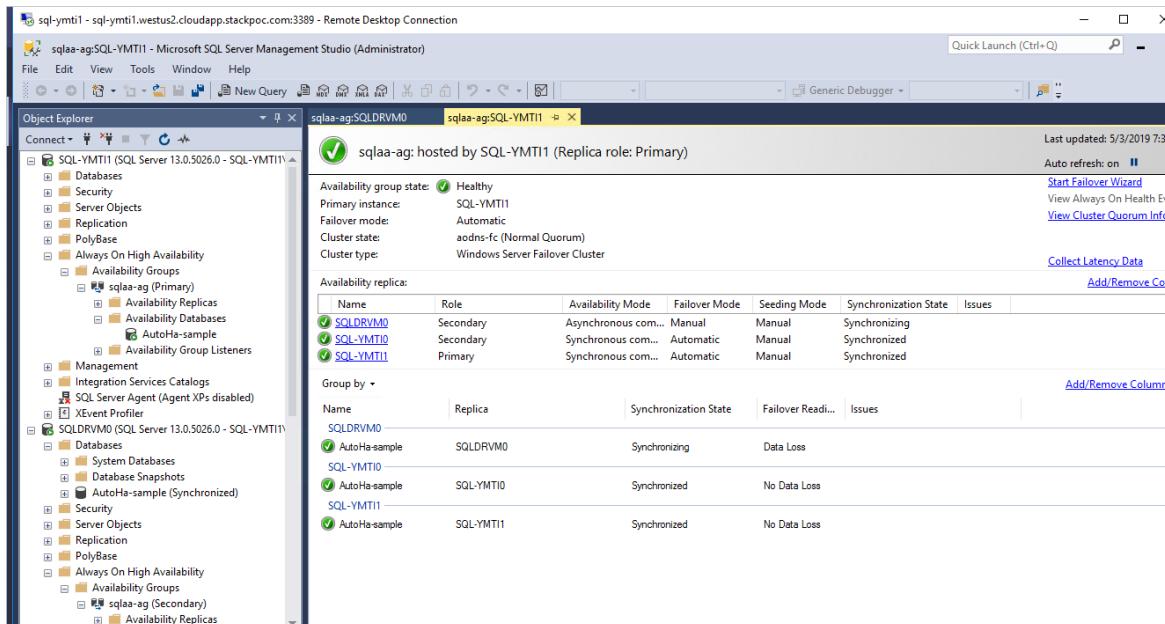
3. Run the deployment. Provide credentials and resource names where needed. HA refers to the Azure Stack Hub where the HA cluster will be deployed. DR refers to the Azure Stack Hub where the DR cluster will be deployed.

```
> .\Deploy-AzureResourceGroup.ps1
-AzureStackApplicationId_HA "applicationIDforHAServicePrincipal"
-AzureStackApplicationSecret_HA "clientSecretforHAServicePrincipal"
-AADTenantName_HA "hatenantname.onmicrosoft.com"
-AzureStackResourceGroup_HA "haresourcegroupname"
-AzureStackArmEndpoint_HA "https://management.haazurestack.com"
-AzureStackSubscriptionId_HA "haSubscriptionId"
-AzureStackApplicationId_DR "applicationIDforDRServicePrincipal"
-AzureStackApplicationSecret_DR "ClientSecretforDRServicePrincipal"
-AADTenantName_DR "drtenantname.onmicrosoft.com"
-AzureStackResourceGroup_DR "drresourcegroupname"
-AzureStackArmEndpoint_DR "https://management.drazerstack.com"
-AzureStackSubscriptionId_DR "drSubscriptionId"
```

4. Type `y` to allow the NuGet provider to be installed, which will kick off the API Profile "2018-03-01-hybrid" modules to be installed.
5. Wait for resource deployment to complete.
6. Once DR resource deployment has completed, exit the container.

```
exit
```

7. Inspect the deployment by viewing the resources in each Azure Stack Hub's portal. Connect to one of the SQL instances on the HA environment and inspect the Availability Group through SQL Server Management Studio (SSMS).



Next steps

- Use SQL Server Management Studio to manually fail over the cluster. See [Perform a Forced Manual Failover of an Always On Availability Group \(SQL Server\)](#).
- Learn more about hybrid cloud apps. See [Hybrid Cloud Solutions](#).
- Use your own data or modify the code to this sample on [GitHub](#).

Direct traffic with a geo-distributed app using Azure and Azure Stack Hub

3/10/2022 • 18 minutes to read • [Edit Online](#)

Learn how to direct traffic to specific endpoints based on various metrics using the geo-distributed apps pattern. Creating a Traffic Manager profile with geographic-based routing and endpoint configuration ensures information is routed to endpoints based on regional requirements, corporate and international regulation, and your data needs.

In this solution, you'll build a sample environment to:

- Create a geo-distributed app.
- Use Traffic Manager to target your app.

Use the geo-distributed apps pattern

With the geo-distributed pattern, your app spans regions. You can default to the public cloud, but some of your users may require that their data remain in their region. You can direct users to the most suitable cloud based on their requirements.

Issues and considerations

Scalability considerations

The solution you'll build with this article isn't to accommodate scalability. However, if used in combination with other Azure and on-premises solutions, you can accommodate scalability requirements. For information on creating a hybrid solution with autoscaling via traffic manager, see [Create cross-cloud scaling solutions with Azure](#).

Availability considerations

As is the case with scalability considerations, this solution doesn't directly address availability. However, Azure and on-premises solutions can be implemented within this solution to ensure high availability for all components involved.

When to use this pattern

- Your organization has international branches requiring custom regional security and distribution policies.
- Each of your organization's offices pulls employee, business, and facility data, which requires reporting activity per local regulations and time zones.
- High-scale requirements are met by horizontally scaling out apps with multiple app deployments within a single region and across regions to handle extreme load requirements.

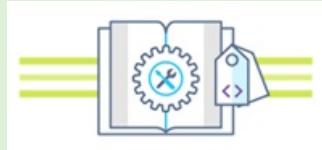
Planning the topology

Before building out a distributed app footprint, it helps to know the following things:

- **Custom domain for the app:** What's the custom domain name that customers will use to access the app? For the sample app, the custom domain name is www.scalableasedemo.com.
- **Traffic Manager domain:** A domain name is chosen when creating an [Azure Traffic Manager profile](#). This name is combined with the *trafficmanager.net* suffix to register a domain entry that's managed by Traffic Manager. For the sample app, the name chosen is *scalable-ase-demo*. As a result, the full domain name that's managed by Traffic Manager is *scalable-ase-demo.trafficmanager.net*.

- **Strategy for scaling the app footprint:** Decide whether the app footprint will be distributed across multiple App Service environments in a single region, multiple regions, or a mix of both approaches. The decision should be based on expectations of where customer traffic will originate and how well the rest of an app's supporting back-end infrastructure can scale. For example, with a 100% stateless app, an app can be massively scaled using a combination of multiple App Service environments per Azure region, multiplied by App Service environments deployed across multiple Azure regions. With 15+ global Azure regions available to choose from, customers can truly build a world-wide hyper-scale app footprint. For the sample app used here, three App Service environments were created in a single Azure region (South Central US).
- **Naming convention for the App Service environments:** Each App Service environment requires a unique name. Beyond one or two App Service environments, it's helpful to have a naming convention to help identify each App Service environment. For the sample app used here, a simple naming convention was used. The names of the three App Service environments are *fe1ase*, *fe2ase*, and *fe3ase*.
- **Naming convention for the apps:** Since multiple instances of the app will be deployed, a name is needed for each instance of the deployed app. With App Service Environment for Power Apps, the same app name can be used across multiple environments. Since each App Service environment has a unique domain suffix, developers can choose to reuse the exact same app name in each environment. For example, a developer could have apps named as follows: *myapp.foo1.p.azurewebsites.net*, *myapp.foo2.p.azurewebsites.net*, *myapp.foo3.p.azurewebsites.net*, and so on. For the app used here, each app instance has a unique name. The app instance names used are *webfrontend1*, *webfrontend2*, and *webfrontend3*.

TIP



Microsoft Azure Stack Hub is an extension of Azure. Azure Stack Hub brings the agility and innovation of cloud computing to your on-premises environment, enabling the only hybrid cloud that allows you to build and deploy hybrid apps anywhere.

The article [Hybrid app design considerations](#) reviews pillars of software quality (placement, scalability, availability, resiliency, manageability, and security) for designing, deploying, and operating hybrid apps. The design considerations assist in optimizing hybrid app design, minimizing challenges in production environments.

Part 1: Create a geo-distributed app

In this part, you'll create a web app.

- Create web apps and publish.
- Add code to Azure Repos.
- Point the app build to multiple cloud targets.
- Manage and configure the CD process.

Prerequisites

An Azure subscription and Azure Stack Hub installation are required.

Geo-distributed app steps

Obtain a custom domain and configure DNS

Update the DNS zone file for the domain. Azure AD can then verify ownership of the custom domain name. Use [Azure DNS](#) for Azure/Microsoft 365/external DNS records within Azure, or add the DNS entry at [a different DNS](#)

registrar.

1. Register a custom domain with a public registrar.
2. Sign in to the domain name registrar for the domain. An approved admin may be required to make the DNS updates.
3. Update the DNS zone file for the domain by adding the DNS entry provided by Azure AD. The DNS entry doesn't change behaviors such as mail routing or web hosting.

Create web apps and publish

Set up Hybrid Continuous Integration/Continuous Delivery (CI/CD) to deploy Web App to Azure and Azure Stack Hub, and auto push changes to both clouds.

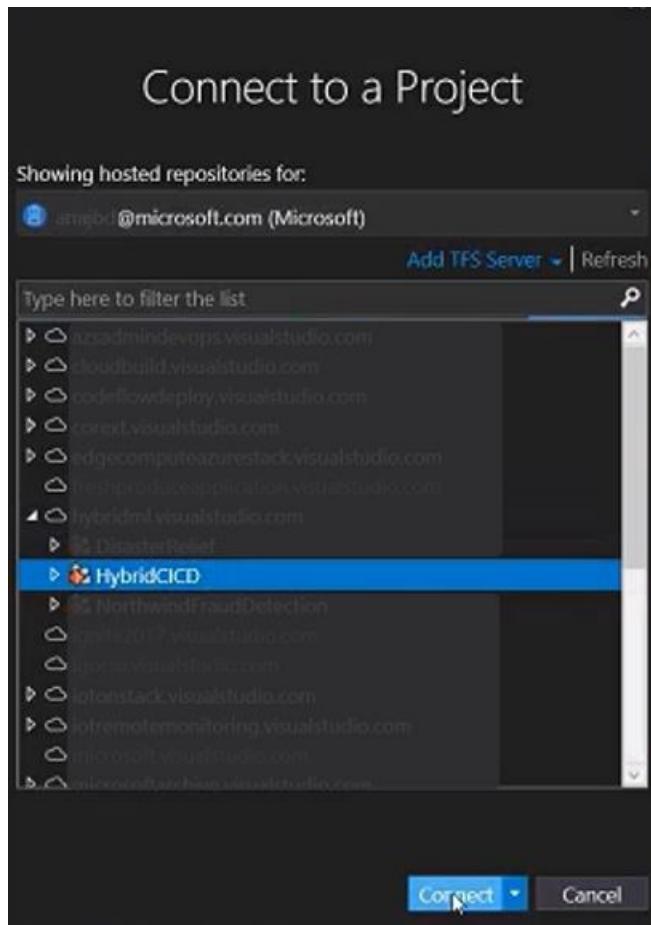
NOTE

Azure Stack Hub with proper images syndicated to run (Windows Server and SQL) and App Service deployment are required. For more information, see [Prerequisites for deploying App Service on Azure Stack Hub](#).

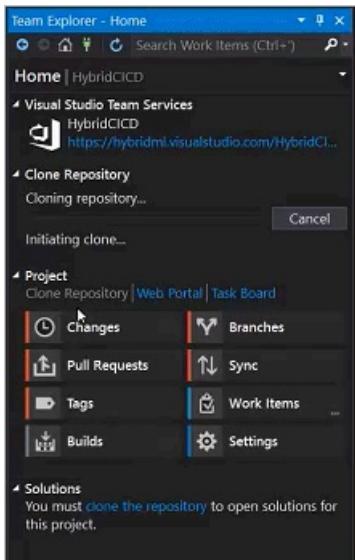
Add Code to Azure Repos

1. Sign in to Visual Studio with an account that has project creation rights on Azure Repos.

CI/CD can apply to both app code and infrastructure code. Use [Azure Resource Manager templates](#) for both private and hosted cloud development.

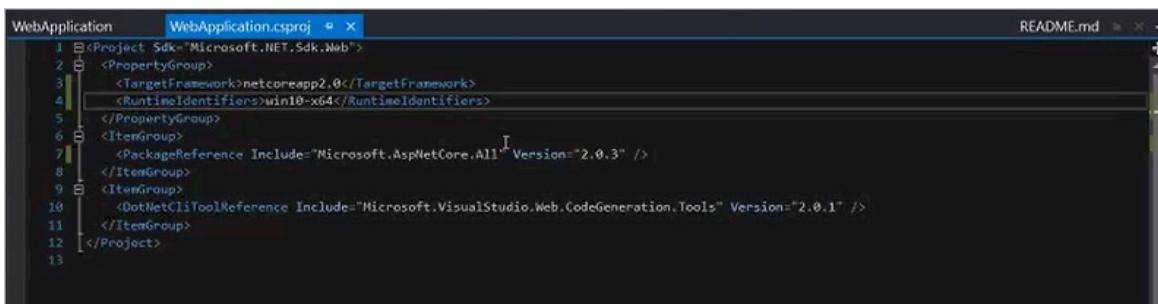


2. Clone the repository by creating and opening the default web app.



Create web app deployment in both clouds

1. Edit the `WebApplication.csproj` file: Select `RuntimeIdentifier` and add `win10-x64`. (See [Self-contained Deployment documentation](#).)

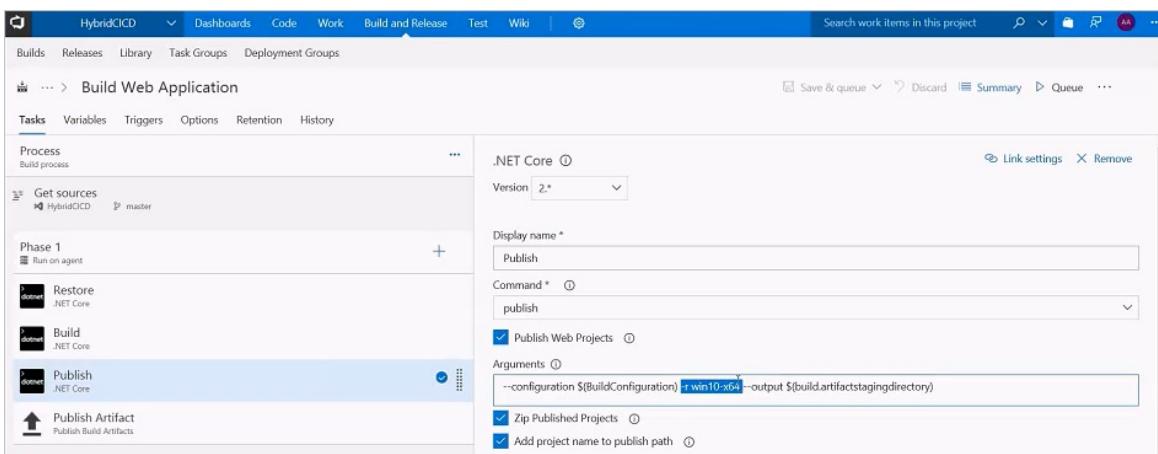


2. Check in the code to Azure Repos using Team Explorer.

3. Confirm that the **application code** has been checked into Azure Repos.

Create the build definition

1. Sign in to [Azure Pipelines](#) to confirm ability to create build definitions.
2. Add `-r win10-x64` code. This addition is necessary to trigger a self-contained deployment with .NET Core.



3. Run the build. The [self-contained deployment](#) build process will publish artifacts that can run on Azure and Azure Stack Hub.

Using an Azure Hosted Agent

Using a hosted agent in Azure Pipelines is a convenient option to build and deploy web apps. Maintenance and upgrades are automatically performed by Microsoft Azure, which enables uninterrupted development, testing,

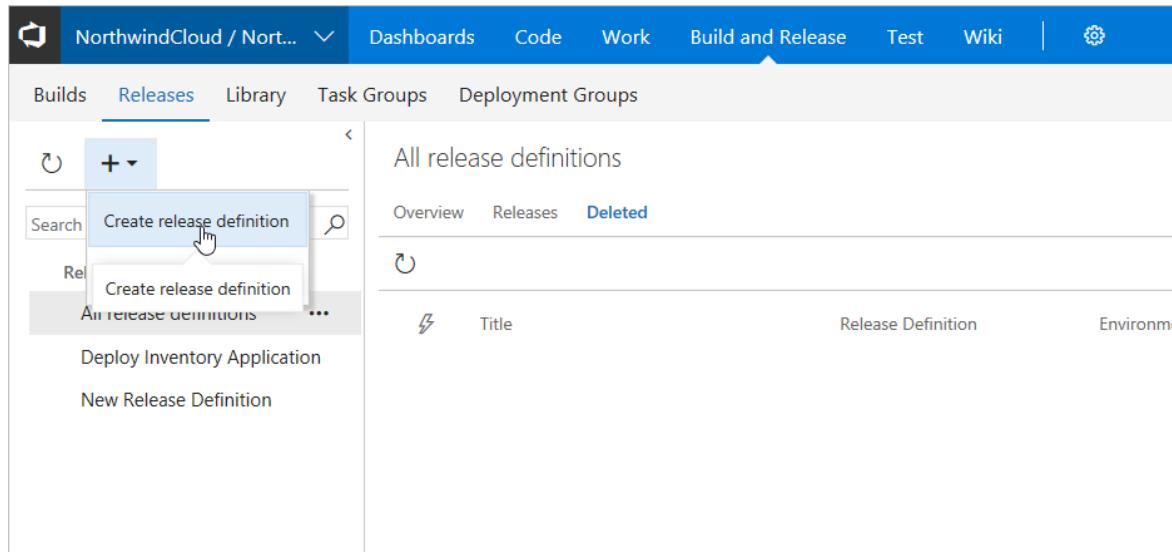
and deployment.

Manage and configure the CD process

Azure DevOps Services provide a highly configurable and manageable pipeline for releases to multiple environments such as development, staging, QA, and production environments; including requiring approvals at specific stages.

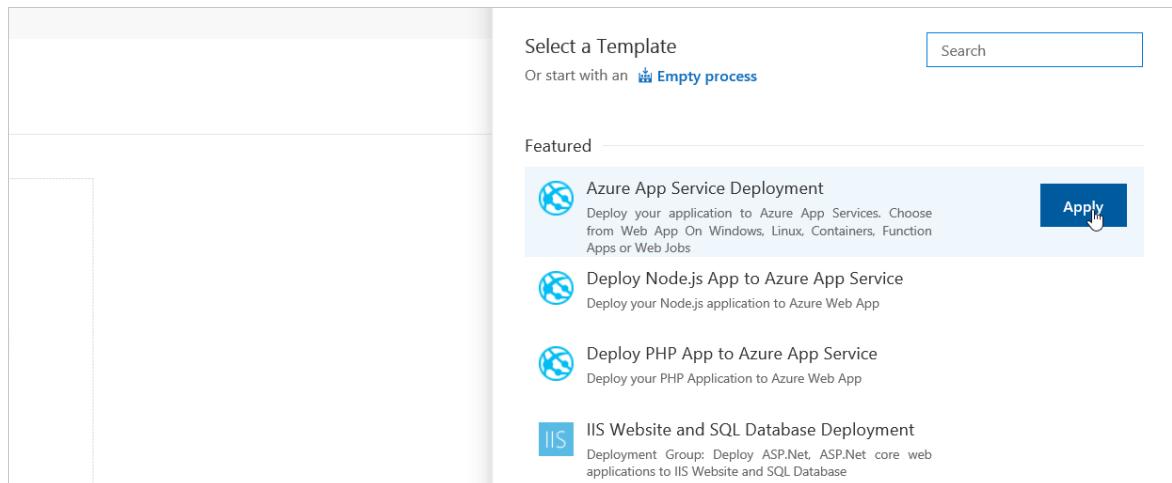
Create release definition

1. Select the plus button to add a new release under the **Releases** tab in the Build and Release section of Azure DevOps Services.



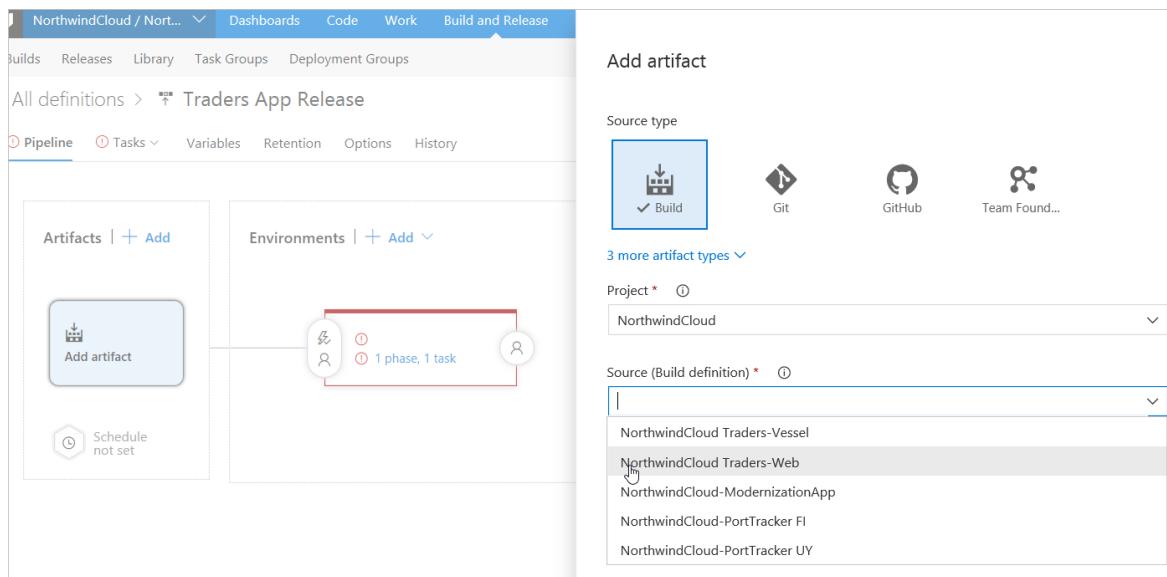
The screenshot shows the Azure DevOps interface for managing releases. The top navigation bar includes 'Dashboards', 'Code', 'Work', 'Build and Release', 'Test', and 'Wiki'. The 'Build and Release' tab is selected. Below it, the 'Releases' tab is active. A modal window is open, showing a list of options: 'Create release definition' (which is highlighted with a mouse cursor), 'Deploy Inventory Application', and 'New Release Definition'. The main area displays 'All release definitions' with tabs for 'Overview', 'Releases', and 'Deleted'. A table lists release definitions with columns for Title, Release Definition, and Environment.

2. Apply the Azure App Service Deployment template.

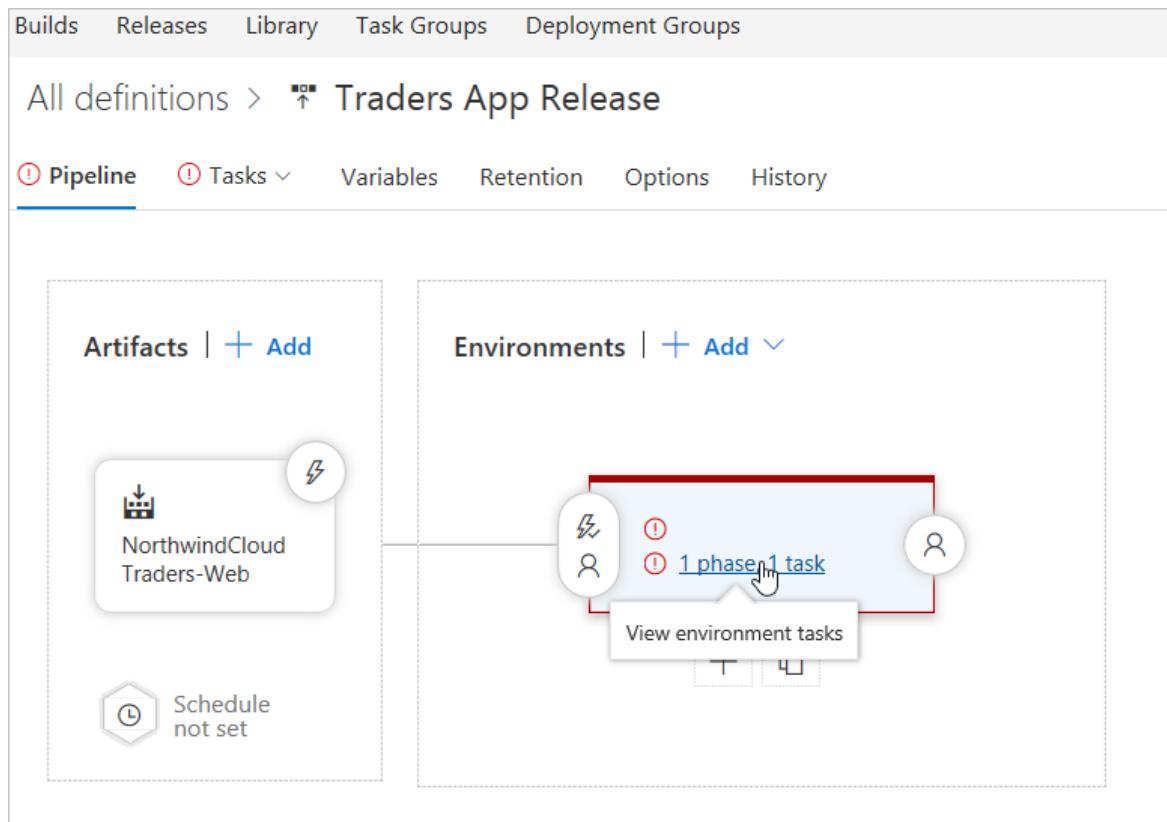


The screenshot shows the 'Select a Template' dialog. It features a search bar and a 'Featured' section. The 'Azure App Service Deployment' template is selected, shown with its icon, name, description, and an 'Apply' button which has a mouse cursor hovering over it. Other templates listed include 'Deploy Node.js App to Azure App Service', 'Deploy PHP App to Azure App Service', and 'IIS Website and SQL Database Deployment'.

3. Under **Add artifact**, add the artifact for the Azure Cloud build app.



4. Under Pipeline tab, select the **Phase, Task** link of the environment and set the Azure cloud environment values.



5. Set the **environment name** and select the **Azure subscription** for the Azure Cloud endpoint.

The screenshot shows the 'Traders App Release' pipeline definition. In the 'Tasks' tab, the 'Deploy Azure App Service' task is selected. The 'Azure subscription' dropdown is open, showing a list of available service connections: AzureStack Traders-Vessel EP, AzureStack PortTracker UY EP, AzureStack PortTracker FI EP, AzureCloud RegulatoryApp EP, AzureStack ModernizationApp Inventory EP, and AzureCloud Traders-Web EP. The 'Available Azure subscriptions' section is also visible.

6. Under App service name, set the required Azure app service name.

The screenshot shows the 'Traders App Release' pipeline definition. In the 'Tasks' tab, the 'Deploy Azure App Service' task is selected. The 'App service name' dropdown is set to 'northwindtraders'. A note below the field states: 'This field is linked to 1 setting in 'Deploy Azure App Service''. The 'Azure subscription' dropdown is also visible.

7. Enter "Hosted VS2017" under Agent queue for Azure cloud hosted environment.

The screenshot shows the 'Traders App Release' pipeline definition. In the 'Tasks' tab, the 'Deploy Azure App Service' task is selected. The 'Agent queue' dropdown is set to 'Hosted VS2017'. The dropdown menu also lists other options like Hosted, Hosted Linux Preview, Hosted macOS Preview, Hosted VS2017, Private, AzureStack - Douglas Fir, and Default (No agents).

8. In Deploy Azure App Service menu, select the valid Package or Folder for the environment. Select OK to folder location.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Deployment process ...

Run on agent Run on agent +

Deploy Azure App Service Azure App Service Deploy ...

northwindtraders

Deploy to slot ⓘ

Virtual application ⓘ

Package or folder * ⓘ \$(System.DefaultWorkingDirectory)/**/*.zip

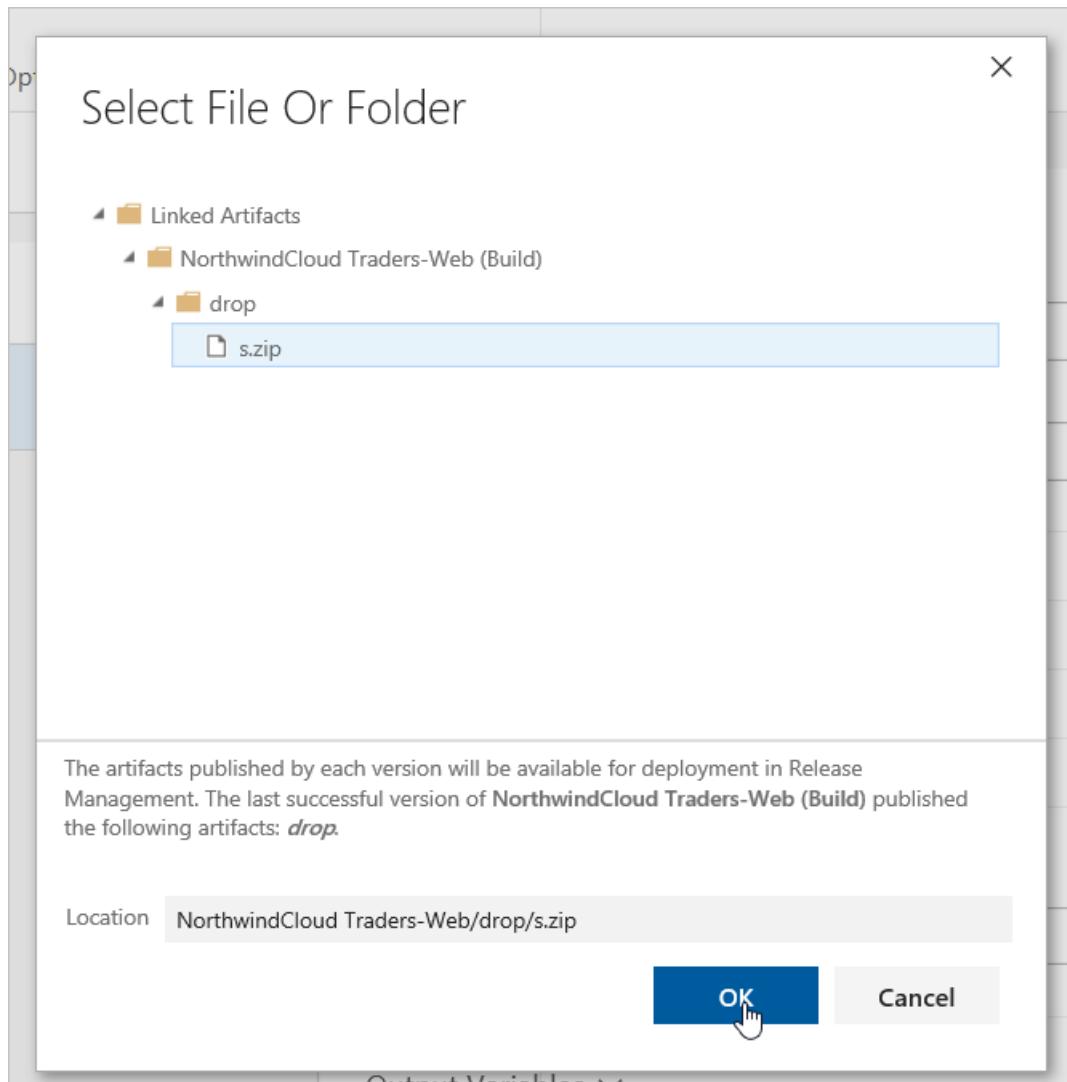
File Transforms & Variable Substitution Options

Additional Deployment Options

Post Deployment Action

Application and Configuration Settings

Output



9. Save all changes and go back to release pipeline.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Deployment process ...

Run on agent Run on agent +

Deploy Azure App Service Azure App Service Deploy ...

Environment name Azure

Parameters ⓘ | Unlink all

Azure subscription * ⚙ | Manage ↗

AzureCloud Traders-Web EP

App type ⚙

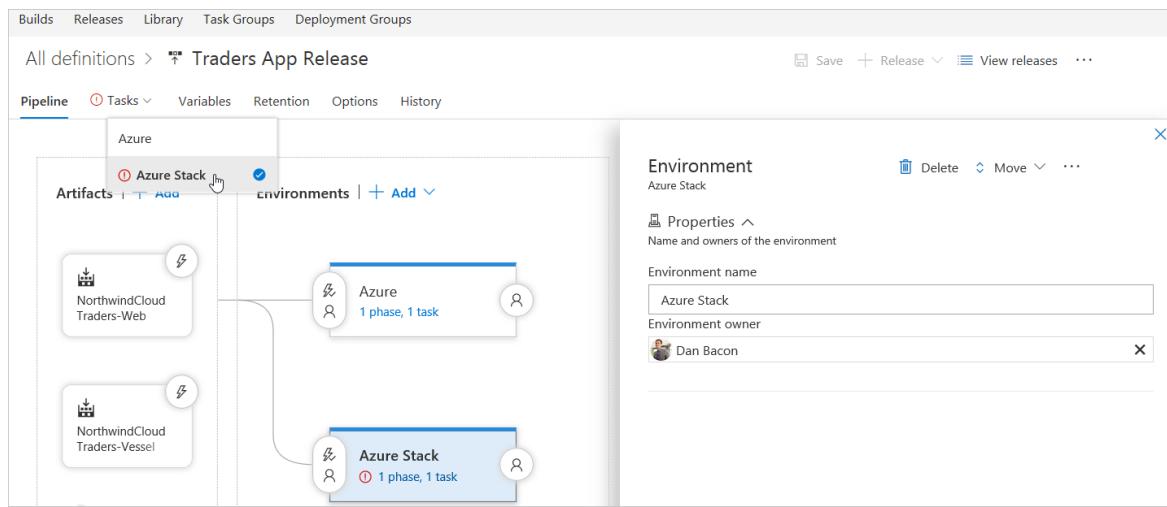
Web App

10. Add a new artifact selecting the build for the Azure Stack Hub app.

11. Add one more environment by applying the Azure App Service Deployment.

12. Name the new environment Azure Stack Hub.

13. Find the Azure Stack Hub environment under Task tab.



14. Select the subscription for the Azure Stack Hub endpoint.

15. Set the Azure Stack Hub web app name as the App service name.

16. Select the Azure Stack Hub agent.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Stack Deployment process

Run on agent Run on agent

Deploy Azure App Service Azure App Service Deploy

Agent phase Agent selection Agent queue

Display name * Run on agent

AzureStack - Douglas Fir Hosted Hosted Hosted Linux Preview Hosted macOS Preview Hosted VS2017 Private AzureStack - Douglas Fir Default (No agents)

Save Release View releases ...

17. Under the Deploy Azure App Service section, select the valid **Package or Folder** for the environment. Select **OK** to folder location.

All definitions > Traders App Release

Pipeline Tasks Variables Retention Options History

Azure Stack Deployment process

Run on agent Run on agent

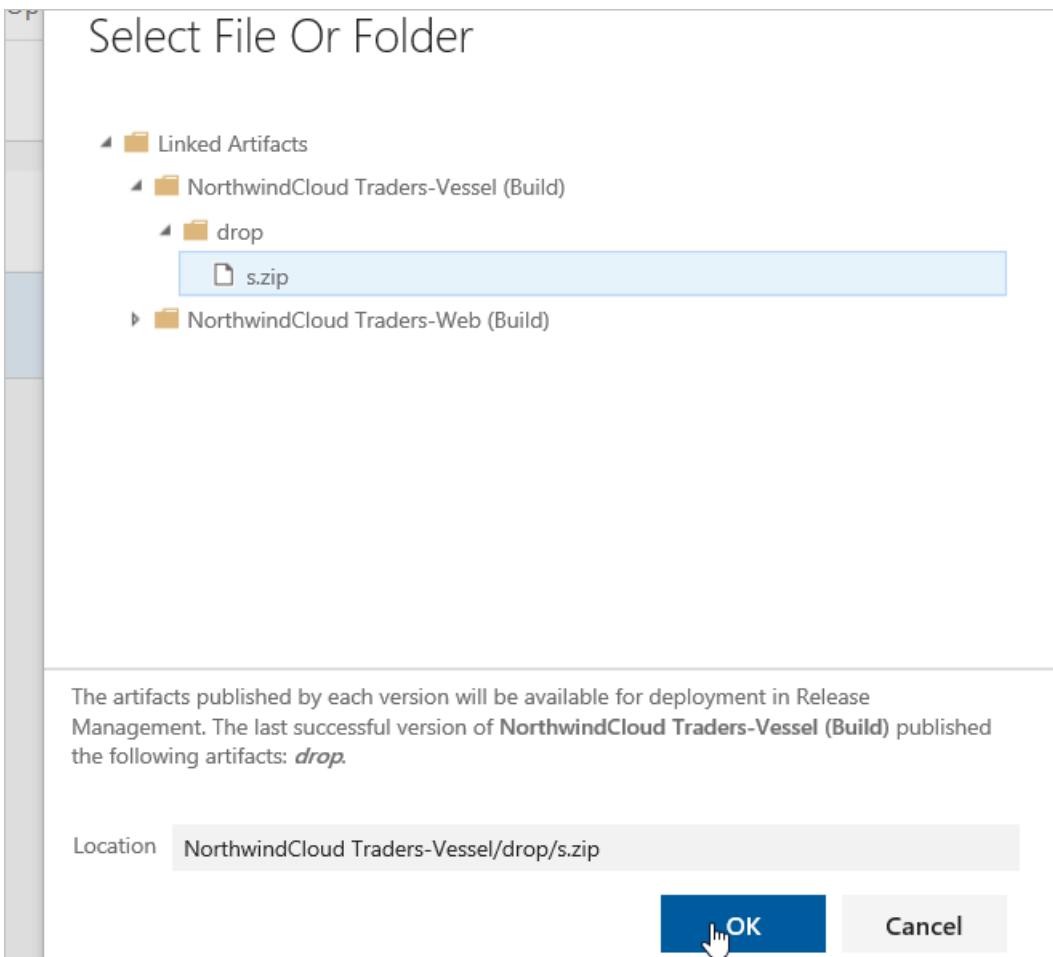
Deploy Azure App Service Azure App Service Deploy

Virtual application Deploy to slot

Package or folder \$System.DefaultWorkingDirectory/**/*.zip

File Transforms & Variable Substitution Options Additional Deployment Options Post Deployment Action Application and Configuration Settings Output App Service URL

Save Release View releases ...

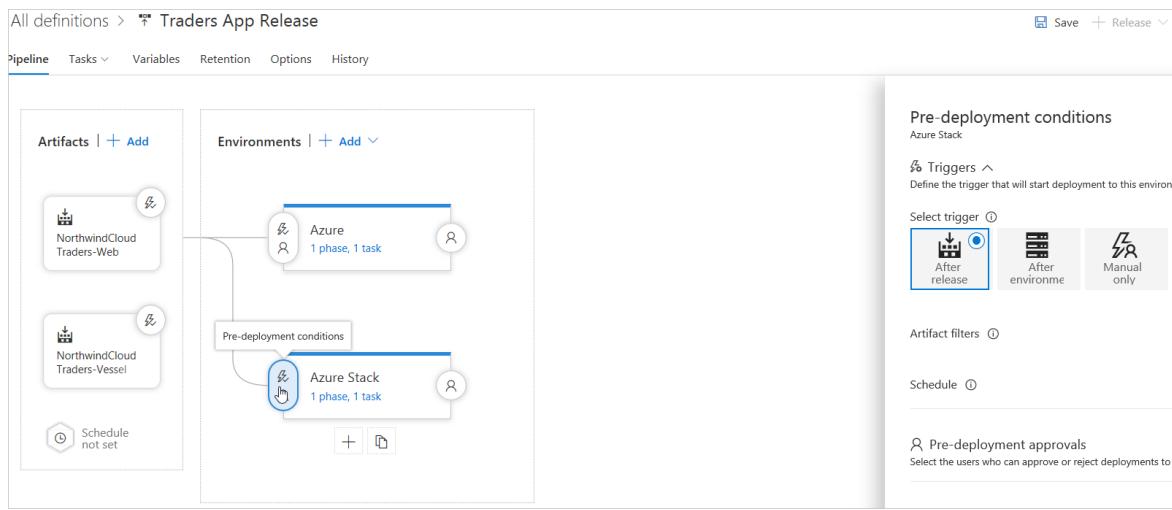


18. Under Variable tab add a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS`, set its value as `true`, and scope to Azure Stack Hub.

Name	Value	Scope
VSTS_ARM_REST_IGNORE_SSL_ERRORS	true	Azure Stack

19. Select the **Continuous** deployment trigger icon in both artifacts and enable the **Continuous** deployment trigger.

20. Select the **Pre-deployment** conditions icon in the Azure Stack Hub environment and set the trigger to **After release**.



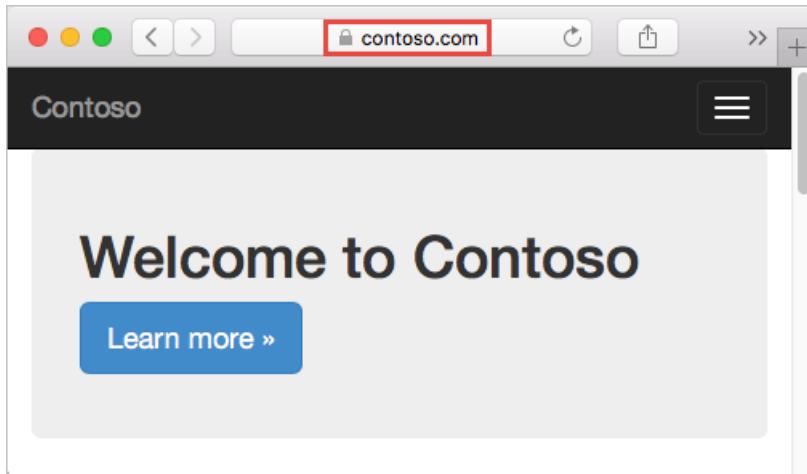
21. Save all changes.

NOTE

Some settings for the tasks may have been automatically defined as [environment variables](#) when creating a release definition from a template. These settings can't be modified in the task settings; instead, the parent environment item must be selected to edit these settings.

Part 2: Update web app options

[Azure App Service](#) provides a highly scalable, self-patching web hosting service.



- Map an existing custom DNS name to Azure Web Apps.
- Use a **CNAME record** and an **A record** to map a custom DNS name to App Service.

Map an existing custom DNS name to Azure Web Apps

NOTE

Use a CNAME for all custom DNS names except a root domain (for example, northwind.com).

To migrate a live site and its DNS domain name to App Service, see [Migrate an active DNS name to Azure App Service](#).

Prerequisites

To complete this solution:

- Create an App Service app, or use an app created for another solution.
- Purchase a domain name and ensure access to the DNS registry for the domain provider.

Update the DNS zone file for the domain. Azure AD will verify ownership of the custom domain name. Use [Azure DNS](#) for Azure/Microsoft 365/external DNS records within Azure, or add the DNS entry at [a different DNS registrar](#).

- Register a custom domain with a public registrar.
- Sign in to the domain name registrar for the domain. (An approved admin may be required to make DNS updates.)
- Update the DNS zone file for the domain by adding the DNS entry provided by Azure AD.

For example, to add DNS entries for northwindcloud.com and www.northwindcloud.com, configure DNS settings for the northwindcloud.com root domain.

NOTE

A domain name may be purchased using the [Azure portal](#). To map a custom DNS name to a web app, the web app's [App Service plan](#) must be a paid tier (**Shared**, **Basic**, **Standard**, or **Premium**).

Create and map CNAME and A records

Access DNS records with domain provider

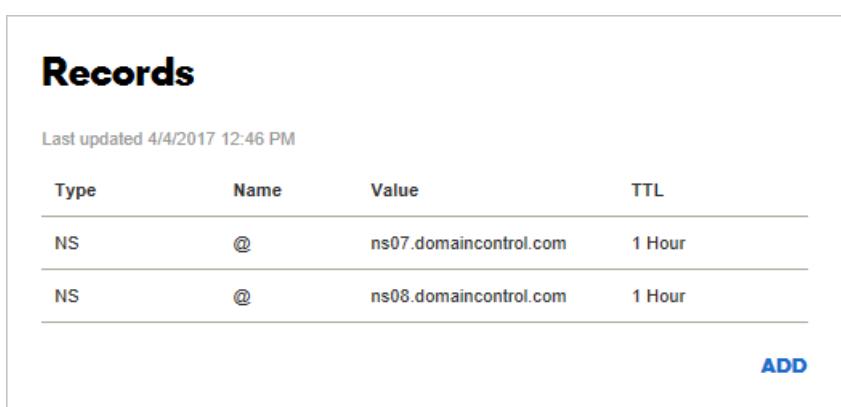
NOTE

Use Azure DNS to configure a custom DNS name for Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

1. Sign in to the website of the main provider.
2. Find the page for managing DNS records. Every domain provider has its own DNS records interface. Look for areas of the site labeled **Domain Name**, **DNS**, or **Name Server Management**.

DNS records page can be viewed in **My domains**. Find the link named **Zone file**, **DNS Records**, or **Advanced configuration**.

The following screenshot is an example of a DNS records page:



The screenshot shows a table titled "Records" with the last update time as 4/4/2017 12:46 PM. The table has four columns: Type, Name, Value, and TTL. There are two rows of data, both of which are NS records pointing to ns07.domaincontrol.com and ns08.domaincontrol.com with a TTL of 1 Hour. An "ADD" button is located at the bottom right of the table.

Type	Name	Value	TTL
NS	@	ns07.domaincontrol.com	1 Hour
NS	@	ns08.domaincontrol.com	1 Hour

ADD

1. In Domain Name Registrar, select **Add** or **Create** to create a record. Some providers have different links to add different record types. Consult the provider's documentation.
2. Add a CNAME record to map a subdomain to the app's default hostname.

For the www.northwindcloud.com domain example, add a CNAME record that maps the name to <app_name>.azurewebsites.net .

After adding the CNAME, the DNS records page looks like the following example:

Records			
Last updated 4/4/2017 3:45 PM			
Type	Name	Value	TTL
CNAME	www	cephalin320170403020701.azurewebsites.net	1 Hour

 [ADD](#)

Enable the CNAME record mapping in Azure

1. In a new tab, sign in to the Azure portal.
2. Go to App Services.
3. Select web app.
4. In the left navigation of the app page in the Azure portal, select **Custom domains**.
5. Select the + icon next to **Add hostname**.
6. Type the fully qualified domain name, like `www.northwindcloud.com`.
7. Select **Validate**.
8. If indicated, add additional records of other types (`A` or `TXT`) to the domain name registrars DNS records. Azure will provide the values and types of these records:
 - a. An A record to map to the app's IP address.
 - b. A TXT record to map to the app's default hostname <app_name>.azurewebsites.net . App Service uses this record only at configuration time to verify custom domain ownership. After verification, delete the TXT record.
9. Complete this task in the domain registrar tab and revalidate until the **Add hostname** button is activated.
10. Make sure that **Hostname record type** is set to **CNAME** ([www.example.com](#) or any subdomain).
11. Select **Add hostname**.
12. Type the fully qualified domain name, like `northwindcloud.com`.
13. Select **Validate**. The **Add** is activated.
14. Make sure that **Hostname record type** is set to **A record** (example.com).
15. **Add hostname**.

It might take some time for the new hostnames to be reflected in the app's **Custom domains** page. Try refreshing the browser to update the data.

HOSTNAMES ASSIGNED TO SITE	
www.northwindcloud.com	...
cephalin320170403020701.azurewebsites.net	

If there's an error, a verification error notification will appear at the bottom of the page.

The screenshot shows a 'Domain Verification' dialog. It has two status indicators: 'Domain ownership' (Error) and 'Hostname availability' (OK). Below these, a message states: 'No CNAME records were found. Please add a CNAME record pointing to cephelin320170403020701.azurewebsites.net'. A red exclamation mark icon is on the left.

NOTE

The above steps may be repeated to map a wildcard domain (*.northwindcloud.com). This allows the addition of any additional subdomains to this app service without having to create a separate CNAME record for each one. Follow the registrar instructions to configure this setting.

Test in a browser

Browse to the DNS name(s) configured earlier (for example, [northwindcloud.com](#) or [www.northwindcloud.com](#)).

Part 3: Bind a custom SSL cert

In this part, we will:

- Bind the custom SSL certificate to App Service.
- Enforce HTTPS for the app.
- Automate SSL certificate binding with scripts.

NOTE

If needed, obtain a customer SSL certificate in the Azure portal and bind it to the web app. For more information, see the [App Service Certificates tutorial](#).

Prerequisites

To complete this solution:

- [Create an App Service app](#).
- [Map a custom DNS name to your web app](#).
- Acquire an SSL certificate from a trusted certificate authority and use the key to sign the request.

Requirements for your SSL certificate

To use a certificate in App Service, the certificate must meet all the following requirements:

- Signed by a trusted certificate authority.
- Exported as a password-protected PFX file.
- Contains private key at least 2048 bits long.
- Contains all intermediate certificates in the certificate chain.

NOTE

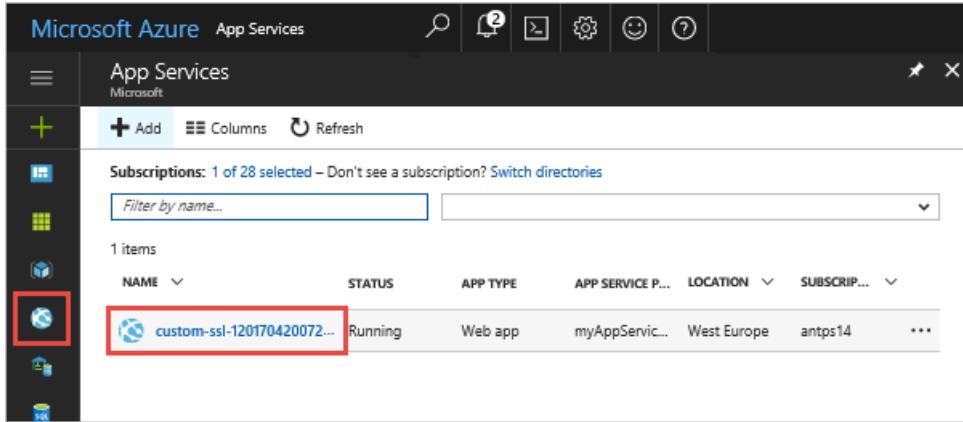
Elliptic Curve Cryptography (ECC) certificates work with App Service but aren't included in this guide. Consult a certificate authority for assistance in creating ECC certificates.

Prepare the web app

To bind a custom SSL certificate to the web app, the [App Service plan](#) must be in the **Basic**, **Standard**, or **Premium** tier.

Sign in to Azure

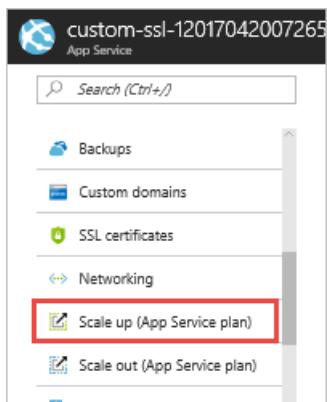
1. Open the [Azure portal](#) and go to the web app.
2. From the left menu, select **App Services**, and then select the web app name.



The screenshot shows the Microsoft Azure App Services dashboard. The top navigation bar includes 'Microsoft Azure' and 'App Services'. Below the navigation are standard icons for search, refresh, and settings. The main area is titled 'Subscriptions: 1 of 28 selected – Don't see a subscription? [Switch directories](#)'. A 'Filter by name...' input field is present. A table lists one item: 'NAME' (custom-ssl-120170420072...), 'STATUS' (Running), 'APP TYPE' (Web app), 'APP SERVICE P...' (myAppService...), 'LOCATION' (West Europe), and 'SUBSCRIP...' (antps14). The entire row for this app is highlighted with a red box.

Check the pricing tier

1. In the left-hand navigation of the web app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.



The screenshot shows the 'custom-ssl-12017042007265' web app settings page. The left sidebar includes 'Backups', 'Custom domains', 'SSL certificates', 'Networking', 'Scale up (App Service plan)', and 'Scale out (App Service plan)'. The 'Scale up (App Service plan)' option is highlighted with a red box.

2. Ensure the web app isn't in the **Free** or **Shared** tier. The web app's current tier is highlighted in a dark blue box.

55.80 USD/MONTH (ESTIMATED)	111.60 USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*
- Shared infrastructure	- Shared infrastructure
1 GB Storage	1 GB Storage Custom domains
0.00 USD/MONTH (ESTIMATED)	9.67 USD/MONTH (ESTIMATED, *PER AP...)
Select	

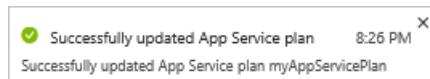
Custom SSL isn't supported in the **Free** or **Shared** tier. To upscale, follow the steps in the next section or the [Choose your pricing tier](#) page and skip to [Upload and bind your SSL certificate](#).

Scale up your App Service plan

1. Select one of the **Basic**, **Standard**, or **Premium** tiers.
2. Select **Select**.

Choose your pricing tier		
Browse the available plans and their features		
USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)	USD/MONTH (ESTIMATED)
B1 Basic	B2 Basic	B3 Basic
1 Core	2 Core	4 Core
1.75 GB RAM	3.5 GB RAM	7 GB RAM
10 GB Storage	10 GB Storage	10 GB Storage
Custom domains	Custom domains	Custom domains
SSL Support SNI SSL Included	SSL Support SNI SSL Included	SSL Support SNI SSL Included
Up to 3 instances Manual scale	Up to 3 instances Manual scale	Up to 3 instances Manual scale
55.80 USD/MONTH (ESTIMATED)	111.60 USD/MONTH (ESTIMATED)	223.20 USD/MONTH (ESTIMATED)
F1 Free	D1 Shared*	
- Shared infrastructure	- Shared infrastructure	
Select		

The scale operation is complete when notification is displayed.



Bind your SSL certificate and merge intermediate certificates

Merge multiple certificates in the chain.

1. Open each certificate you received in a text editor.
2. Create a file for the merged certificate called *mergedcertificate.crt*. In a text editor, copy the content of each certificate into this file. The order of your certificates should follow the order in the certificate chain, beginning with your certificate and ending with the root certificate. It looks like the following example:

```
-----BEGIN CERTIFICATE-----  
  
<your entire Base64 encoded SSL certificate>  
  
-----END CERTIFICATE-----  
  
-----BEGIN CERTIFICATE-----  
  
<The entire Base64 encoded intermediate certificate 1>  
  
-----END CERTIFICATE-----  
  
-----BEGIN CERTIFICATE-----  
  
<The entire Base64 encoded intermediate certificate 2>  
  
-----END CERTIFICATE-----  
  
-----BEGIN CERTIFICATE-----  
  
<The entire Base64 encoded root certificate>  
  
-----END CERTIFICATE-----
```

Export certificate to PFX

Export the merged SSL certificate with the private key generated by the certificate.

A private key file is created via OpenSSL. To export the certificate to PFX, run the following command and replace the placeholders <private-key-file> and <merged-certificate-file> with the private key path and the merged certificate file:

```
openssl pkcs12 -export -out myserver.pfx -inkey <private-key-file> -in <merged-certificate-file>
```

When prompted, define an export password for uploading your SSL certificate to App Service later.

When IIS or **Certreq.exe** are used to generate the certificate request, install the certificate to a local machine and then [export the certificate to PFX](#).

Upload the SSL certificate

1. Select **SSL settings** in the left navigation of the web app.
2. Select **Upload Certificate**.
3. In **PFX Certificate File**, select PFX file.
4. In **Certificate password**, type the password created when exporting the PFX file.
5. Select **Upload**.

Add certificate X

Secure your app's custom domain with HTTPS [Close]

Type Private Public

* PFX Certificate File
 Select a file [Open]

Certificate password

When App Service finishes uploading the certificate, it appears in the SSL settings page.

 SSL

Certificates must be associated with your application before you can use them to create a binding. You can upload a certificate you purchased externally, or import an App Service Certificate.

 Import App Service Certificate  Upload Certificate

NAME	EXPIRATION	THUMBPRINT
www.northwindcloud.com		...

SSL bindings

 Add binding

HOST NAME	CERTIFICATE	SSL TYPE
No results		

Bind your SSL certificate

1. In the SSL bindings section, select **Add binding**.

NOTE

If the certificate has been uploaded, but doesn't appear in domain name(s) in the Hostname dropdown, try refreshing the browser page.

2. In the Add SSL Binding page, use the drop downs to select the domain name to secure and the certificate to use.
3. In SSL Type, select whether to use **Server Name Indication (SNI)** or IP-based SSL.
 - **SNI-based SSL:** Multiple SNI-based SSL bindings may be added. This option allows multiple SSL certificates to secure multiple domains on the same IP address. Most modern browsers (including Internet Explorer, Chrome, Firefox, and Opera) support SNI (find more comprehensive browser support information at [Server Name Indication](#)).
 - **IP-based SSL:** Only one IP-based SSL binding may be added. This option allows only one SSL certificate to secure a dedicated public IP address. To secure multiple domains, secure them all using the same SSL certificate. IP-based SSL is the traditional option for SSL binding.
4. Select Add Binding.

The screenshot shows the 'SSL' blade in the Azure portal. At the top, there's a green shield icon with a lock and the word 'SSL'. Below it, a message says: 'Certificates must be associated with your application before you can use them to create a binding. You can upload a certificate you purchased externally, or import an App Service Certificate.' There are two buttons: 'Import App Service Certificate' with a downward arrow icon and 'Upload Certificate' with an upward arrow icon. A table lists existing certificates: contoso.com, www.c... (Expiration: 4/20/2018), and an ellipsis button. Below this is a section titled 'SSL bindings' with a 'Add binding' button. A table shows a single binding: www.northwindcloud.com (Certificate: IP Based SSL, SSL Type: IP Based SSL), also with an ellipsis button. A red box highlights the 'www.northwindcloud.com' row.

HOST NAME	CERTIFICATE	SSL TYPE
www.northwindcloud.com	IP Based SSL	...

When App Service finishes uploading the certificate, it appears in the **SSL bindings** sections.

The screenshot shows the Azure portal's SSL configuration page. At the top left is a green shield icon with a lock, followed by the text "SSL". Below this, a message states: "Certificates must be associated with your application before you can use them to create a binding. You can upload a certificate you purchased externally, or import an App Service Certificate." Two buttons are present: "Import App Service Certificate" with a downward arrow icon and "Upload Certificate" with an upward arrow icon. A table lists existing certificates: "contoso.com, www.c..." with expiration on "4/20/2018" and a three-dot ellipsis button. Below this is a section titled "SSL bindings" with a "Add binding" button featuring a plus sign. A table lists a single binding: "HOST NAME" "www.northwindcloud.com", "CERTIFICATE" "IP Based SSL", and a three-dot ellipsis button. The entire interface is framed by a thick black border.

Remap the A record for IP SSL

If IP-based SSL isn't used in the web app, skip to [Test HTTPS for your custom domain](#).

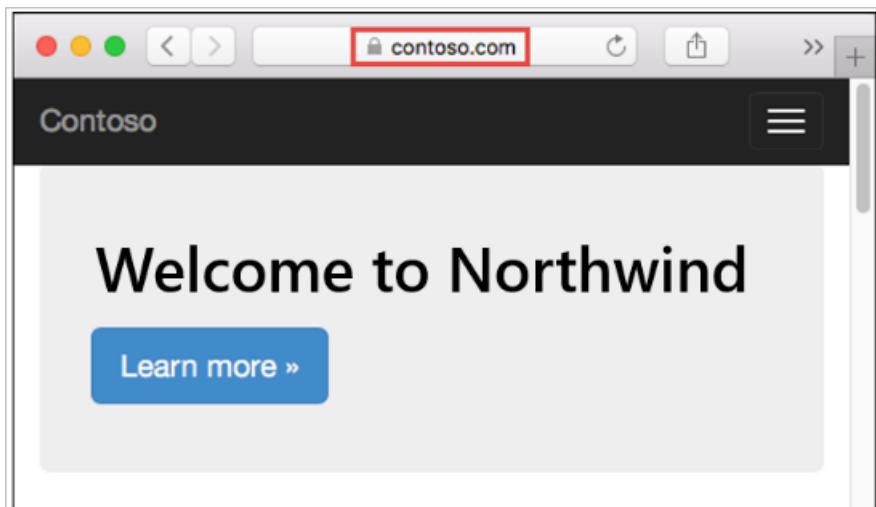
By default, the web app uses a shared public IP address. When the certificate is bound with IP-based SSL, App Service creates a new and dedicated IP address for the web app.

When an A record is mapped to the web app, the domain registry must be updated with the dedicated IP address.

The [Custom domain](#) page is updated with the new, dedicated IP address. Copy this [IP address](#), then remap the [A record](#) to this new IP address.

Test HTTPS

In different browsers, go to <https://<your.custom.domain>> to ensure the web app is served.



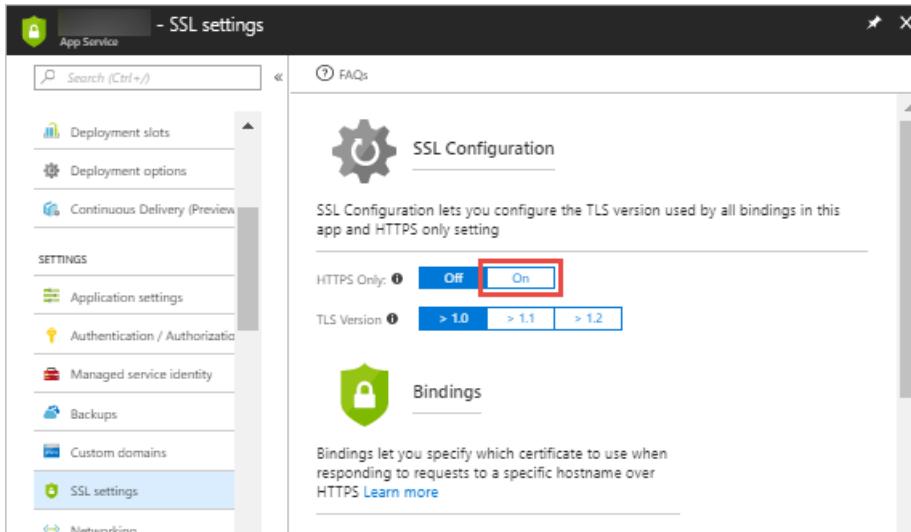
NOTE

If certificate validation errors occur, a self-signed certificate may be the cause, or intermediate certificates may have been left off when exporting to the PFX file.

Enforce HTTPS

By default, anyone can access the web app using HTTP. All HTTP requests to the HTTPS port may be redirected.

In the web app page, select **SSL settings**. Then, in **HTTPS Only**, select **On**.



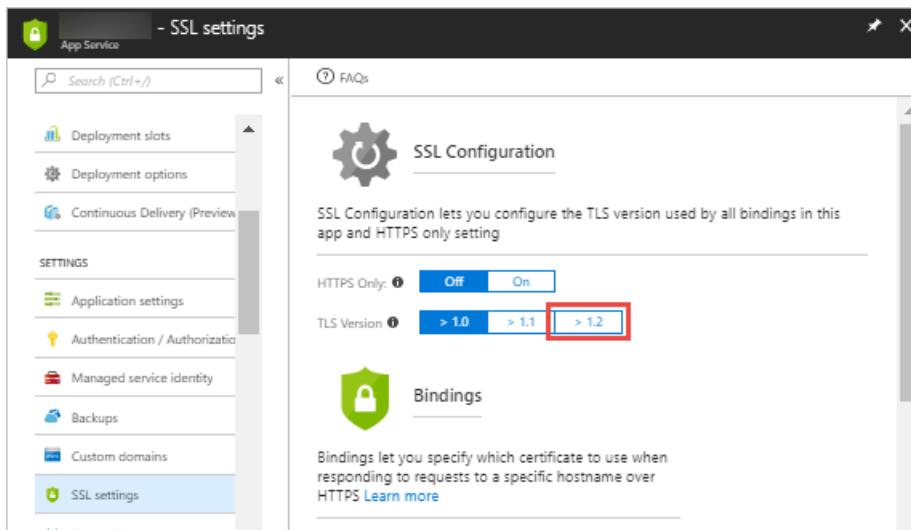
When the operation is complete, go to any of the HTTP URLs that point to the app. For example:

- https://<app_name>.azurewebsites.net
- <https://northwindcloud.com>
- <https://www.northwindcloud.com>

Enforce TLS 1.1/1.2

The app allows **TLS 1.0** by default, which is no longer considered secure by industry standards (like [PCI DSS](#)). To enforce higher TLS versions, follow these steps:

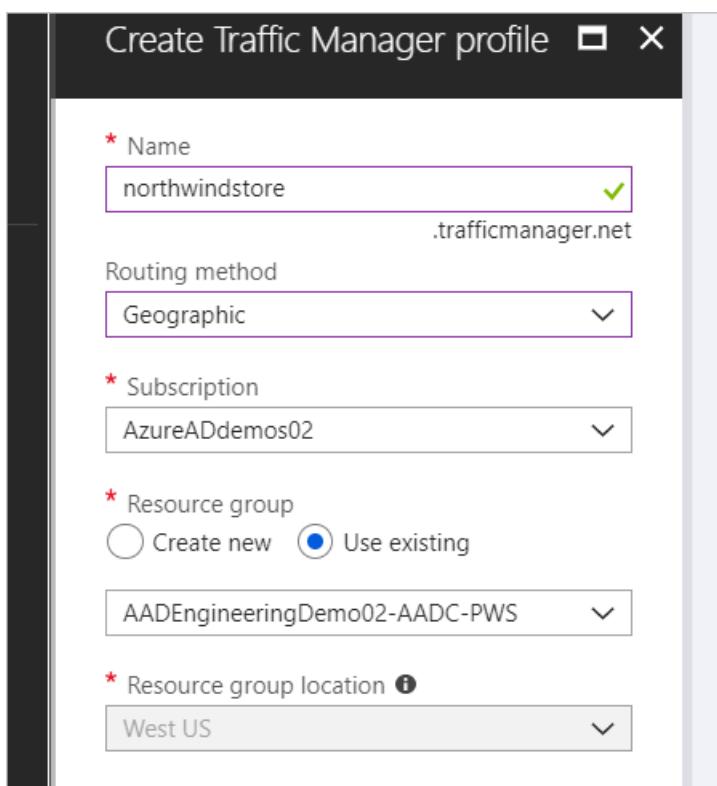
1. In the web app page, in the left navigation, select **SSL settings**.
2. In **TLS version**, select the minimum TLS version.



Create a Traffic Manager profile

1. Select **Create a resource > Networking > Traffic Manager profile > Create**.

2. In the **Create Traffic Manager profile**, complete as follows:
 - a. In **Name**, provide a name for the profile. This name needs to be unique within the traffic manager.net zone and results in the DNS name, trafficmanager.net, which is used to access the Traffic Manager profile.
 - b. In **Routing method**, select the **Geographic routing method**.
 - c. In **Subscription**, select the subscription under which to create this profile.
 - d. In **Resource Group**, create a new resource group to place this profile under.
 - e. In **Resource group location**, select the location of the resource group. This setting refers to the location of the resource group and has no impact on the Traffic Manager profile deployed globally.
 - f. Select **Create**.
 - g. When the global deployment of the Traffic Manager profile is complete, it's listed in the respective resource group as one of the resources.



Add Traffic Manager endpoints

1. In the portal search bar, search for the **Traffic Manager profile** name created in the preceding section and select the traffic manager profile in the displayed results.
2. In **Traffic Manager profile**, in the **Settings** section, select **Endpoints**.
3. Select **Add**.
4. Adding the Azure Stack Hub Endpoint.
5. For **Type**, select **External endpoint**.
6. Provide a **Name** for this endpoint, ideally the name of the Azure Stack Hub.
7. For fully qualified domain name (**FQDN**), use the external URL for the Azure Stack Hub Web App.
8. Under Geo-mapping, select a region/continent where the resource is located. For example, **Europe**.
9. Under the Country/Region drop-down that appears, select the country that applies to this endpoint. For

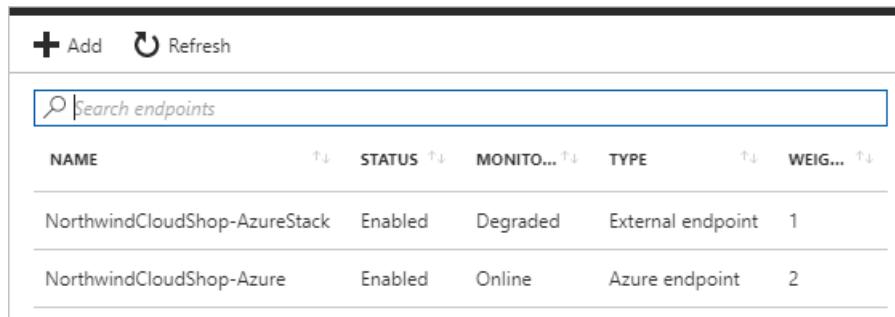
example, **Germany**.

10. Keep **Add as disabled** unchecked.
11. Select **OK**.
12. Adding the Azure Endpoint:
 - a. For **Type**, select **Azure endpoint**.
 - b. Provide a **Name** for the endpoint.
 - c. For **Target resource type**, select **App Service**.
 - d. For **Target resource**, select **Choose an app service** to show the listing of the Web Apps under the same subscription. In **Resource**, pick the App service used as the first endpoint.
13. Under Geo-mapping, select a region/continent where the resource is located. For example, **North America/Central America/Caribbean**.
14. Under the Country/Region drop-down that appears, leave this spot blank to select all of the above regional grouping.
15. Keep **Add as disabled** unchecked.
16. Select **OK**.

NOTE

Create at least one endpoint with a geographic scope of All (World) to serve as the default endpoint for the resource.

17. When the addition of both endpoints is complete, they're displayed in **Traffic Manager profile** along with their monitoring status as **Online**.



The screenshot shows a table listing two endpoints. The columns are NAME, STATUS, MONITO..., TYPE, and WEIG... (Weight). The first endpoint, 'NorthwindCloudShop-AzureStack', is of type 'External endpoint' with weight 1. The second endpoint, 'NorthwindCloudShop-Azure', is of type 'Azure endpoint' with weight 2. Both endpoints are currently online.

NAME	STATUS	MONITO...	TYPE	WEIG...
NorthwindCloudShop-AzureStack	Enabled	Degraded	External endpoint	1
NorthwindCloudShop-Azure	Enabled	Online	Azure endpoint	2

Global Enterprise relies on Azure geo-distribution capabilities

Directing data traffic via Azure Traffic Manager and geography-specific endpoints enables global enterprises to adhere to regional regulations and keep data compliant and secure, which is crucial to the success of local and remote business locations.

Next steps

- To learn more about Azure Cloud Patterns, see [Cloud Design Patterns](#).

Conditional Access for Zero Trust

3/10/2022 • 2 minutes to read • [Edit Online](#)

The articles in this section provide a design and framework for implementing [Zero Trust](#) principles by using Conditional Access to control access to cloud services. The guidance is based on years of experience with helping customers control access to their resources.

The framework presented here represents a structured approach that you can use to get a good balance between security and usability while ensuring that user access is controlled.

The guidance suggests a structured approach for helping to secure access that's based on personas. It also includes a breakdown of suggested personas and defines the Conditional Access policies for each persona.

Intended audience

This guidance is intended for individuals who:

- Design security and identity solutions to control access to Azure protected resources.
- Maintain solutions after they're delivered.

The intended audience has a basic working knowledge of Azure Active Directory (Azure AD) and a general understanding of multi-factor authentication, Conditional Access, identity, and security concepts.

Knowledge in the following areas is also recommended:

- Microsoft Endpoint Manager
- Azure AD identity management
- Azure AD Conditional Access and multi-factor authentication for guest users (B2B)
- Azure AD security policies and resource protection
- The B2B invitation process

Requirements

Every company has different requirements and security policies. When you create an architecture and follow this suggested framework for Conditional Access, you need to take your company's requirements into account. The guidance includes principles that are related to Zero Trust that you can use as input when you create an architecture. You can then address specific company requirements and policies and adjust the architecture accordingly.

For example, a company might have these requirements:

- All access must be protected by at least two factors.
- No data on unmanaged devices.
- No guest access allowed.
- Access to cloud services must be based on passwordless authentication.

Conditional Access guidance

This section includes the following articles:

- [Conditional Access design principles and dependencies](#) provides recommended principles that, together with your company's requirements, serve as input to the suggested persona-based architecture.

- [Conditional Access architecture and personas](#) introduces the persona-based approach for structuring Conditional Access policies. It also provides suggested personas that you can use as a starting point.
- [Conditional Access framework and policies](#) provides specific details on how to structure and name Conditional Access policies that are based on the personas.

Next steps

- [Learning path: Implement and manage identity and access](#)
- [What is Conditional Access?](#)
- [Common Conditional Access policies](#)

Related resources

- [Conditional Access design principles and dependencies](#)
- [Conditional Access architecture and personas](#)
- [Conditional Access framework and policies](#)
- [Azure Active Directory IDaaS in security operations](#)

Conditional Access framework and policies

3/10/2022 • 9 minutes to read • [Edit Online](#)

This article provides a framework for implementing a persona-based Conditional Access architecture, like the one described in [Conditional Access Zero Trust architecture](#). In this article, there are details on how to form and name the Conditional Access policies. There's also a starting point for creating policies.

If you don't use a framework like the one provided here, including a naming standard, policies tend to be created over time by different people in an ad-hoc way. This can result in policies that overlap. If the person who created a policy is no longer available, it can be difficult for others to know the purpose of the policy.

Following a structured framework makes it easier to understand the policies. It also makes it easier to cover all scenarios and avoid conflicting policies that are difficult to troubleshoot.

Naming conventions

A properly defined naming convention helps you and your colleagues understand the purpose of a policy, which enables easier policy management and troubleshooting. Your naming convention should fit the framework you use to structure your policies.

The recommended naming policy is based on personas, policy types, and apps. It looks like this:

<CANumber>-<Persona>-<PolicyType>-<App>-<Platform>-<GrantControl>-
<OptionalDescription>

The components of this name are described here:

NAMING COMPONENT	DESCRIPTION/EXAMPLE
CA Number	Used to quickly identify Policy Type scope and order. Example: CA001-CA099.
Persona	Global, Admins, Internals, External, GuestUsers, GuestAdmins, Microsoft365ServiceAccounts, AzureServiceAccounts, CorpServiceAccounts.
Policy Type	BaseProtection, AppProtection, DataProtection, IdentityProtection, AttackSurfaceReduction, Compliance.
App	AllApps, O365 (for all Office 365 services), EXO (for Exchange Online).
Platform	AnyPlatform, Unknown, WindowsPhone, macOS, iOS, Android.
Grant Control	Block, ADHJ, Compliant, Unmanaged (where unmanaged is specified in the device state condition).
Description	Optional description of the purpose of the policy.

Numbering scheme

To make administration easy, we suggest this numbering scheme:

PERSONA GROUP	NUMBER ALLOCATION
CA-Persona-Global	CA001-CA099
CA-Persona-Internals	CA100-CA199
CA-Persona-Admins	CA200-CA299
CA-Persona-Externals	CA300-CA399
CA-Persona-GuestUsers	CA400-CA499
CA-Persona-GuestAdmins	CA500-CA599
CA-Persona-M365ServiceAccounts	CA600-CA699
CA-Persona-AzureServiceAccounts	CA700-CA799
CA-Persona-CorpServiceAccounts	CA800-CA899
CA-Persona-WorkloadIdentities	CA900-CA999
CA-Persona-Developers	CA1000-CA1099

Policy types

These are the recommended policy types:

POLICY TYPE	DESCRIPTION/EXAMPLES
BaseProtection	<p>For each persona, there's a baseline protection that's covered by this policy type. For users on managed devices, this typically is known user and known device. For external guests, it might be known user and multi-factor authentication.</p> <p>The base protection is the default policy for all apps for users in the given persona. If a specific app should have a different policy, exclude that app from the base protection policy and add an explicit policy that targets only that app.</p> <p>Example: Assume the base protection for Internals is to require known user and known device for all cloud apps, but you want to allow Outlook on the web from any device. You could exclude Exchange Online from the base protection policy and add a separate policy for Exchange Online, where you require known device OR multi-factor authentication.</p>
IdentityProtection	<p>On top of the base protection for each persona, you can have Conditional Access policies that relate to identity.</p> <p>Examples: Block legacy authentication, require extra multi-factor authentication for high user or sign-in risk, require known device for multi-factor authentication registration.</p>

POLICY TYPE	DESCRIPTION/EXAMPLES
DataProtection	<p>This policy type indicates delta policies that protect data as an extra layer on top of the base protection.</p> <p>Examples:</p> <ul style="list-style-type: none"> • App protection policies for iOS and Android that you can use to encrypt data on a phone and that provide improved protection of that data. (App protection policies also include app protection.) • Session policies where Azure Information Protection helps secure data during download if the data is sensitive.
AppProtection	<p>This policy type is another addition to the base protection.</p> <p>Examples:</p> <ul style="list-style-type: none"> • Assume you want to allow web access to Exchange Online from any device. You could exclude Exchange from the base policy and create a new explicit policy for access to Exchange, where, for example, you allow only read-only access to Exchange Online. • Assume you require multi-factor authentication for Microsoft Endpoint Manager enrollment. You could exclude Intune Endpoint Manager enrollment from the base policy and add an app protection policy that requires multi-factor authentication for Endpoint Manager enrollment.
AttackSurfaceReduction	<p>The purpose of this type of policy is to mitigate various attacks.</p> <p>Examples:</p> <ul style="list-style-type: none"> • If an access attempt comes from an unknown platform, it might be an attempt to bypass Conditional Access policies in which you require a specific platform. You can block requests from unknown platforms to mitigate this risk. • Block access to Office 365 services for Azure Administrators or block access to an app for all users if the app is a known to be bad.
Compliance	You can use a compliance policy to require a user to view the terms of use for guests who access customer services.

App

The following table describes the App component of a policy name:

APP NAME	DESCRIPTION/EXAMPLES
----------	----------------------

APP NAME	DESCRIPTION/EXAMPLES
AllApps	AllApps indicates that all cloud apps are targeted in the Conditional Access policy. All endpoints are covered, including those that support Conditional Access and those that don't. In some scenarios, targeting all apps doesn't work well. We recommend targeting all apps in the base policy so that all endpoints are covered by that policy. New apps that appear in Azure AD will also automatically adhere to the policy.
<AppName>	<p><AppName> is a placeholder for the name of an app that the policy addresses. Avoid making the name too long.</p> <p>Example names:</p> <ul style="list-style-type: none"> • EXO for Exchange Online • SPO for SharePoint Online

Platform type

The following table describes the Platform component of a policy name:

PLATFORM TYPE	DESCRIPTION
AnyPlatform	The policy targets any platform. You typically configure this by selecting Any Device . (In Conditional Access policies, both the word <i>platform</i> and the word <i>device</i> are used.)
iOS	The policy targets Apple iOS platforms.
Android	The policy targets Google Android platforms.
WindowsPhone	The policy targets Windows Phone platforms.
macOS	The policy targets the macOS platforms
iOSAndroid	The policy targets both iOS and Android platforms.
Unknown	The policy targets any platform not listed previously. You typically configure this by including Any Device and excluding all the individual platforms.

Grant control types

The following table describes the Grant Control component of a policy name:

GRANT TYPE	DESCRIPTION/EXAMPLES
Multi-factor authentication	The policy requires multi-factor authentication.
Compliant	The policy requires a compliant device, as determined by Endpoint Manager, so the device needs to be managed by Endpoint Manager.

GRANT TYPE	DESCRIPTION/EXAMPLES
CompliantOrAADHJ	The policy requires a compliant device OR a Hybrid Azure AD joined device. A standard company computer that's domain joined is also Hybrid Azure AD joined. Mobile phones and Windows 10 computers that are co-managed or Azure AD joined can be compliant.
CompliantAndAADHJ	The policy requires a device that's compliant AND Hybrid Azure AD joined.
MFAOrCompliant	The policy requires a compliant device OR multi-factor authentication if it's not compliant.
MFAAndCompliant	The policy requires a compliant device AND multi-factor authentication.
MFAOrAADHJ	The policy requires a Hybrid Azure AD joined computer OR multi-factor authentication if it's not a Hybrid Azure AD joined computer.
MFAAndAADHJ	The policy requires a Hybrid Azure AD joined computer AND multi-factor authentication.
Unmanaged	The policy targets devices that aren't known by Azure AD. For example, you can use this grant type to allow access to Exchange Online from any device.

Named locations

We recommend that you define these standard locations for use in Conditional Access policies:

- **Trusted IPs / Internal networks.** These IP subnets represent locations and networks that have physical access restrictions or other controls in place, like computer system management, network-level authentication, or intrusion detection. These locations are more secure, so Conditional Access enforcement can be relaxed. Consider whether Azure or other datacenter locations (IPs) should be included in this location or have their own named locations.
- **Citrix-trusted IPs.** If you have Citrix on-premises, it might be useful to configure separate outgoing IPv4 addresses for the Citrix farms, if you need to be able to connect to cloud services from Citrix sessions. In that case, you can exclude those locations from Conditional Access policies if you need to.
- **Zscaler locations, if applicable.** Computers have a ZPA agent installed and forward all traffic to the internet to or through Zscaler cloud. So it's worth defining Zscaler source IPs in Conditional Access and requiring all requests from non-mobile devices to go through Zscaler.
- **Countries with which to allow business.** It can be useful to divide countries into two location groups: one that represents areas of the world where employees typically work and one that represents other locations. This allows you to apply additional controls to requests that originate from outside the areas where your organization normally operates.
- **Locations where multi-factor authentication might be difficult or impossible.** In some scenarios, requiring multi-factor authentication could make it difficult for employees to do their work. For example, staff might not have the time or opportunity to respond to frequent multi-factor authentication challenges. Or, in some locations, RF screening or electrical interference can make the use of mobile devices difficult. Typically, you'd use other controls in these locations, or they might be trusted.

Location-based access controls rely on the source IP of a request to determine the location of the user at the

time of the request. It's not easy to perform spoofing on the public internet, but protection afforded by network boundaries might be considered less relevant than it once was. We don't recommend relying solely on location as a condition for access. But for some scenarios it might be the best control that you can use, like if you're securing access from a service account from on-premises that's used in a non-interactive scenario.

Recommended Conditional Access policies

We've created a spreadsheet that contains recommended Conditional Access policies. You can [download the spreadsheet here](#).

Use the suggested policies as a starting point.

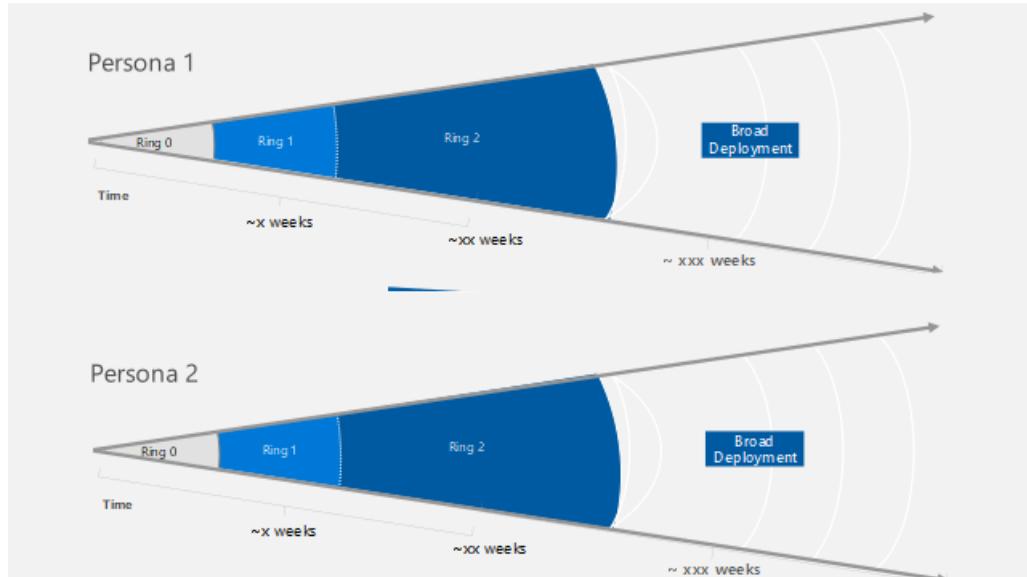
Your policies will probably change over time to accommodate use cases that are important to your business. Here are a few examples of scenarios that might require changes:

- Implement read-only access to Exchange Online for employees using any unmanaged device based on multi-factor authentication, app protection, and an approved client app.
- Implement information protection to ensure that sensitive information isn't downloaded without improved protection provided by Azure Information Protection.
- Provide protection against copy and paste by guests.

Conditional Access guidance

Now that you have a starter set of Conditional Access policies, you need to deploy them in a controlled and phased way. We suggest that you use a deployment model.

Here's one approach:



The idea is to first deploy policies to a small number of users within one persona group. You can use an associated Azure AD group called Persona Ring 0 for this deployment. After you verify that everything works, you change the assignment to a group, Persona Ring 1, that has more members, and so on.

You then enable the policies by using the same ring-based approach until all policies are deployed.

You typically manage the members of ring 0 and ring 1 manually. A ring 2 or 3 group that contains hundreds or even thousands of users could be managed via a dynamic group that's based on a percentage of the users in a given persona.

The use of rings as part of a deployment model isn't just for initial deployment. You can also use it when new policies or changes to existing policies are required.

With a finished deployment, you should also design and implement the monitoring controls that were discussed in the [Conditional Access principles](#).

In addition to automating the initial deployment, you might want to automate changes to policies by using CI/CD pipelines. You could use Microsoft365DSC for this automation.

Next steps

- [Learning path: Implement and manage identity and access](#)
- [Conditional Access policies](#)

Related resources

- [Conditional Access overview](#)
- [Conditional Access design principles and dependencies](#)
- [Conditional Access design based on Zero Trust and personas](#)
- [Azure Active Directory IDaaS in security operations](#)

Work with claims-based identities

3/10/2022 • 5 minutes to read • [Edit Online](#)



Sample code

Claims in Azure AD

When a user signs in, Azure AD sends an ID token that contains a set of claims about the user. A claim is simply a piece of information, expressed as a key/value pair. For example, `email = bob@contoso.com`. Claims have an issuer (in this case, Azure AD), which is the entity that authenticates the user and creates the claims. You trust the claims because you trust the issuer. (Conversely, if you don't trust the issuer, don't trust the claims!)

At a high level:

1. The user authenticates.
2. The Identity Provider (IDP) sends a set of claims.
3. The app normalizes or augments the claims (optional).
4. The app uses the claims to make authorization decisions.

In OpenID Connect, the set of claims that you get is controlled by the [scope parameter](#) of the authentication request. However, Azure AD issues a limited set of claims through OpenID Connect; see [Supported Token and Claim Types](#). If you want more information about the user, you'll need to use the Azure AD Graph API.

Here are some of the claims from Azure AD that an app might typically care about:

CLAIM TYPE IN ID TOKEN	DESCRIPTION
aud	Who the token was issued for. This will be the application's client ID. Generally, you shouldn't need to worry about this claim, because the middleware automatically validates it. Example: <code>"91464657-d17a-4327-91f3-2ed99386406f"</code>
groups	A list of Azure AD groups of which the user is a member. Example: <code>["93e8f556-8661-4955-87b6-890bc043c30f", "fc781505-18ef-4a31-a7d5-7d931d7b857e"]</code>
iss	The issuer of the OIDC token. Example: <code>https://sts.windows.net/b9bd2162-77ac-4fb2-8254-5c36e9c0a9c4/</code>
name	The user's display name. Example: <code>"Alice A."</code>
oid	The object identifier for the user in Azure AD. This value is the immutable and non-reusable identifier of the user. Use this value, not email, as a unique identifier for users; email addresses can change. If you use the Azure AD Graph API in your app, object ID is that value used to query profile information. Example: <code>"59f9d2dc-995a-4ddf-915e-b3bb314a7fa4"</code>

CLAIM TYPE IN ID TOKEN	DESCRIPTION
roles	A list of app roles for the user. Example: ["SurveyCreator"]
tid	Tenant ID. This value is a unique identifier for the tenant in Azure AD. Example: "b9bd2162-77ac-4fb2-8254-5c36e9c0a9c4"
unique_name	A human readable display name of the user. Example: "alice@contoso.com"
upn	User principal name. Example: "alice@contoso.com"

This table lists the claim types as they appear in the ID token. In ASP.NET Core, the OpenID Connect middleware converts some of the claim types when it populates the Claims collection for the user principal:

- oid > `http://schemas.microsoft.com/identity/claims/objectidentifier`
- tid > `http://schemas.microsoft.com/identity/claims/tenantid`
- unique_name > `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`
- upn > `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn`

Claims transformations

During the authentication flow, you might want to modify the claims that you get from the IDP. In ASP.NET Core, you can perform claims transformation inside of the **AuthenticationValidated** event from the OpenID Connect middleware. (See [Authentication events](#).)

Any claims that you add during **AuthenticationValidated** are stored in the session authentication cookie. They don't get pushed back to Azure AD.

Here are some examples of claims transformation:

- **Claims normalization**, or making claims consistent across users. This is particularly relevant if you are getting claims from multiple IDPs, which might use different claim types for similar information.
- Add **default claim values** for claims that aren't present (for example, assigning a user to a default role). In some cases this can simplify authorization logic.
- Add **custom claim types** with application-specific information about the user. For example, you might store some information about the user in a database. You could add a custom claim with this information to the authentication ticket. The claim is stored in a cookie, so you only need to get it from the database once per login session. On the other hand, you also want to avoid creating excessively large cookies, so you need to consider the trade-off between cookie size versus database lookups.

After the authentication flow is complete, the claims are available in `HttpContext.User`. At that point, you should treat them as a read-only collection—for example, using them to make authorization decisions.

Issuer validation

In OpenID Connect, the issuer claim ("iss") identifies the IDP that issued the ID token. Part of the OIDC authentication flow is to verify that the issuer claim matches the actual issuer. The OIDC middleware handles this for you.

In Azure AD, the issuer value is unique per AD tenant (`https://sts.windows.net/<tenantID>`). Therefore, an application should do another check, to make sure the issuer represents a tenant that is allowed to sign in to the

app.

For a single-tenant application, you can just check that the issuer is your own tenant. In fact, the OIDC middleware does this automatically by default. In a multitenant app, you need to allow for multiple issuers, corresponding to the different tenants. Here is a general approach to use:

- When a tenant signs up, store the tenant and the issuer in your user DB.
- Whenever a user signs in, look up the issuer in the database. If the issuer isn't found, it means that tenant hasn't signed up. You can redirect them to a sign-up page.
- You could also block certain tenants; for example, for customers that didn't pay their subscription.

For a more detailed discussion, see [Sign-up and tenant onboarding in a multitenant application](#).

Using claims for authorization

With claims, a user's identity is no longer a monolithic entity. For example, a user might have an email address, phone number, birthday, gender, etc. Maybe the user's IDP stores all of this information. But when you authenticate the user, you'll typically get a subset of these as claims. In this model, the user's identity is simply a bundle of claims. When you make authorization decisions about a user, you will look for particular sets of claims. In other words, the question "Can user X perform action Y" ultimately becomes "Does user X have claim Z".

Here are some basic patterns for checking claims.

- To check that the user has a particular claim with a particular value:

```
if (User.HasClaim(ClaimTypes.Role, "Admin")) { ... }
```

This code checks whether the user has a Role claim with the value "Admin". It correctly handles the case where the user has no Role claim or multiple Role claims.

The **ClaimTypes** class defines constants for commonly used claim types. However, you can use any string value for the claim type.

- To get a single value for a claim type, when you expect there to be at most one value:

```
string email = User.FindFirst(ClaimTypes.Email)?.Value;
```

- To get all the values for a claim type:

```
IEnumerable<Claim> groups = User.FindAll("groups");
```

For more information, see [Role-based and resource-based authorization in multitenant applications](#).

[Next](#)

Tenant sign-up and onboarding

3/10/2022 • 6 minutes to read • [Edit Online](#)



[Sample code](#)

This article describes how to implement a *sign-up* process in a multitenant application, which allows a customer to sign up their organization for your application.

There are several reasons to implement a sign-up process:

- Allow an AD admin to consent for the customer's entire organization to use the application.
- Collect credit card payment or other customer information.
- Perform any one-time per-tenant setup needed by your application.

Admin consent and Azure AD permissions

In order to authenticate with Azure AD, an application needs access to the user's directory. At a minimum, the application needs permission to read the user's profile. The first time that a user signs in, Azure AD shows a consent page that lists the permissions being requested. By clicking **Accept**, the user grants permission to the application.

By default, consent is granted on a per-user basis. Every user who signs in sees the consent page. However, Azure AD also supports *admin consent*, which allows an AD administrator to consent for an entire organization.

When the admin consent flow is used, the consent page states that the AD admin is granting permission on behalf of the entire tenant:

The screenshot shows the Azure AD Admin Consent screen for an application named "Survey". The app publisher website is listed as "localhost". The page asks for permission to "Sign in and read user profile". It shows that the user is signed in as "alice@contoso.com (admin)". A note states that if agreed, the app will have access for all users in the organization. At the bottom are "Accept" and "Cancel" buttons.

After the admin clicks **Accept**, other users within the same tenant can sign in, and Azure AD will skip the consent screen.

Only an AD administrator can give admin consent, because it grants permission on behalf of the entire organization. If a non-administrator tries to authenticate with the admin consent flow, Azure AD displays an error:

Additional technical information:

Correlation ID: 662b80ea-fb8b-4fe9-a368-02f597854249

Timestamp: 2015-11-12 23:28:46Z

AADSTS90093: This operation can only be performed by an administrator. Sign out and sign in as an administrator or contact one of your organization's administrators.

If the application requires additional permissions at a later point, the customer will need to sign up again and consent to the updated permissions.

Implementing tenant sign-up

For the [Tailspin Surveys](#) application, we defined several requirements for the sign-up process:

- A tenant must sign up before users can sign in.
- Sign-up uses the admin consent flow.
- Sign-up adds the user's tenant to the application database.
- After a tenant signs up, the application shows an onboarding page.

In this section, we'll walk through our implementation of the sign-up process. It's important to understand that "sign up" versus "sign in" is an application concept. During the authentication flow, Azure AD does not inherently know whether the user is in process of signing up. It's up to the application to keep track of the context.

When an anonymous user visits the Surveys application, the user is shown two buttons, one to sign in, and one to "enroll your company" (sign up).



The screenshot shows the Tailspin Surveys application interface. At the top, there is a dark header bar with the Tailspin logo on the left and a 'Sign in' button on the right, both of which are highlighted with red boxes. Below the header, the main content area has a light gray background. In the center, the text 'Welcome to Tailspin!' is displayed in a large, bold, dark font. Below this, there is a blue rectangular button with white text that reads 'Enroll your company in Tailspin!'. This button is also highlighted with a red box. To the right of the button, a note in parentheses says '(You need to be an admin of the Azure Active Directory of your company to enroll.)'. Further down the page, under the heading 'Published Surveys', there is a light blue horizontal bar containing the text 'No surveys have been published.'

These buttons invoke actions in the `AccountController` class.

The `signIn` action returns a `ChallengeResult`, which causes the OpenID Connect middleware to redirect to the authentication endpoint. This is the default way to trigger authentication in ASP.NET Core.

```
[AllowAnonymous]
public IActionResult SignIn()
{
    return new ChallengeResult(
        OpenIdConnectDefaults.AuthenticationScheme,
        new AuthenticationProperties
        {
            IsPersistent = true,
            RedirectUri = Url.Action("SignInCallback", "Account")
        });
}
```

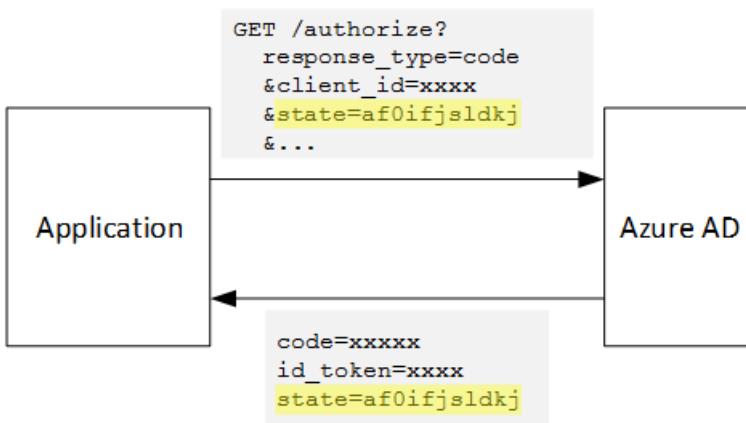
Now compare the `SignUp` action:

```
[AllowAnonymous]
public IActionResult SignUp()
{
    var state = new Dictionary<string, string> { { "signup", "true" } };
    return new ChallengeResult(
        OpenIdConnectDefaults.AuthenticationScheme,
        new AuthenticationProperties(state)
        {
            RedirectUri = Url.Action(nameof(SignUpCallback), "Account")
        });
}
```

Like `SignIn`, the `SignUp` action also returns a `ChallengeResult`. But this time, we add a piece of state information to the `AuthenticationProperties` in the `ChallengeResult`:

- `signup`: A Boolean flag, indicating that the user has started the sign-up process.

The state information in `AuthenticationProperties` gets added to the OpenID Connect `state` parameter, which round trips during the authentication flow.



After the user authenticates in Azure AD and gets redirected back to the application, the authentication ticket contains the state. We are using this fact to make sure the "signup" value persists across the entire authentication flow.

Adding the admin consent prompt

In Azure AD, the admin consent flow is triggered by adding a "prompt" parameter to the query string in the authentication request:

```
/authorize?prompt=admin_consent&...
```

The Surveys application adds the prompt during the `RedirectToIdentityProvider` event. This event is called right before the middleware redirects to the authentication endpoint.

```
public override Task RedirectToIdentityProvider(RedirectContext context)
{
    if (context.IsSignUp())
    {
        context.ProtocolMessage.Prompt = "admin_consent";
    }

    _logger.RedirectToIdentityProvider();
    return Task.FromResult(0);
}
```

Setting `ProtocolMessage.Prompt` tells the middleware to add the "prompt" parameter to the authentication request.

Note that the prompt is only needed during sign-up. Regular sign-in should not include it. To distinguish between them, we check for the `signup` value in the authentication state. The following extension method checks for this condition:

```
internal static bool IsSignUp(this AuthenticationProperties properties)
{
    Guard.ArgumentNotNull(properties, nameof(properties));

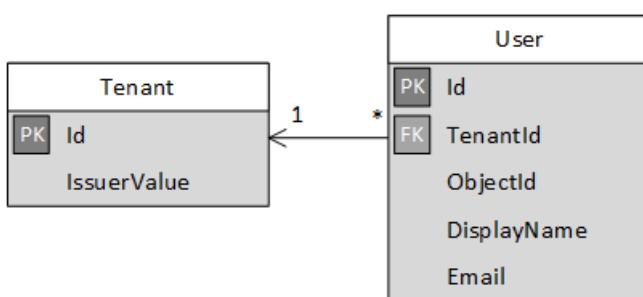
    string signupValue = string.Empty;
    // Check the HTTP context and convert to string
    if ((properties == null) ||
        (!properties.Items.TryGetValue("signup", out signupValue)))
    {
        return false;
    }

    // We have found the value, so see if it's valid
    bool isSignUp;
    if (!bool.TryParse(signupValue, out isSignUp))
    {
        // The value for signup is not a valid boolean, throw
        throw new InvalidOperationException($"'{signupValue}' is an invalid boolean value");
    }

    return isSignUp;
}
```

Registering a tenant

The Surveys application stores some information about each tenant and user in the application database.



In the Tenant table, `IssuerValue` is the value of the issuer claim for the tenant. For Azure AD, this is <https://sts.windows.net/<tenantID>> and gives a unique value per tenant.

When a new tenant signs up, the Surveys application writes a tenant record to the database. This happens inside the `AuthenticationValidated` event. (Don't do it before this event, because the ID token won't be validated yet, so you can't trust the claim values. See [Authentication](#).)

Here is the relevant code from the Surveys application:

```
public override async Task TokenValidated(TokenValidatedContext context)
{
    var principal = context.Principal;
    var userId = principal.GetObjectIdentifierValue();
    var tenantManager = context.HttpContext.RequestServices.GetService<TenantManager>();
    var userManager = context.HttpContext.RequestServices.GetService<UserManager>();
    var issuerValue = context.SecurityToken.Issuer;
    _logger.AuthenticationValidated(userId, issuerValue);

    // Normalize the claims first.
    NormalizeClaims(principal);
    var tenant = await tenantManager.FindByIssuerValueAsync(issuerValue);

    if (context.Properties.IsSigningUp())
    {
        if (tenant == null)
        {
            tenant = await SignUpTenantAsync(context, tenantManager)
                .ConfigureAwait(false);
        }

        // In this case, we need to go ahead and set up the user signing us up.
        await CreateOrUpdateUserAsync(context.Ticket, userManager, tenant)
            .ConfigureAwait(false);
    }
    else
    {
        if (tenant == null)
        {
            _logger.UnregisteredUserSignInAttempted(userId, issuerValue);
            throw new SecurityTokenValidationException($"Tenant {issuerValue} is not registered");
        }

        await CreateOrUpdateUserAsync(context.Principal, userManager, tenant)
            .ConfigureAwait(false);
    }
}
```

This code does the following:

1. Check if the tenant's issuer value is already in the database. If the tenant has not signed up, `FindByIssuerValueAsync` returns null.
2. If the user is signing up:
 - a. Add the tenant to the database (`SignUpTenantAsync`).
 - b. Add the authenticated user to the database (`CreateOrUpdateUserAsync`).
3. Otherwise complete the normal sign-in flow:
 - a. If the tenant's issuer was not found in the database, it means the tenant is not registered, and the customer needs to sign up. In that case, throw an exception to cause the authentication to fail.
 - b. Otherwise, create a database record for this user, if there isn't one already (`CreateOrUpdateUserAsync`).

Here is the `SignUpTenantAsync` method that adds the tenant to the database.

```

private async Task<Tenant> SignUpTenantAsync(TokenValidatedContext context, TenantManager tenantManager)
{
    Guard.ArgumentNotNull(context, nameof(context));
    Guard.ArgumentNotNull(tenantManager, nameof(tenantManager));

    var principal = context.Principal;
    var issuerValue = principal.GetIssuerValue();
    var tenant = new Tenant
    {
        IssuerValue = issuerValue,
        Created = DateTimeOffset.UtcNow
    };

    try
    {
        await tenantManager.CreateAsync(tenant)
            .ConfigureAwait(false);
    }
    catch(Exception ex)
    {
        _logger.SignUpTenantFailed(principal.GetObjectIdentifierValue(), issuerValue, ex);
        throw;
    }

    return tenant;
}

```

Here is a summary of the entire sign-up flow in the Surveys application:

1. The user clicks the **Sign Up** button.
2. The `AccountController.SignUp` action returns a challenge result. The authentication state includes "signup" value.
3. In the `RedirectToAuthenticationEndpoint` event, add the `admin_consent` prompt.
4. The OpenID Connect middleware redirects to Azure AD and the user authenticates.
5. In the `AuthenticationValidated` event, look for the "signup" state.
6. Add the tenant to the database.

[Next](#)

Application roles

3/10/2022 • 5 minutes to read • [Edit Online](#)



[Sample code](#)

Application roles are used to assign permissions to users. For example, the [Tailspin Surveys](#) application defines the following roles:

- Administrator. Can perform all CRUD operations on any survey that belongs to that tenant.
- Creator. Can create new surveys.
- Reader. Can read any surveys that belong to that tenant.

You can see that roles ultimately get translated into permissions, during [authorization](#). But the first question is how to assign and manage roles. We identified three main options:

- [Azure AD App Roles](#)
- [Azure AD security groups](#)
- [Application role manager](#).

Roles using Azure AD App Roles

This is the approach that we used in the Tailspin Surveys app.

In this approach, The SaaS provider defines the application roles by adding them to the application manifest. After a customer signs up, an admin for the customer's AD directory assigns users to the roles. When a user signs in, the user's assigned roles are sent as claims.

NOTE

If the customer has Azure AD Premium, the admin can assign a security group to a role, and user members of the group will inherit the app role. This is a convenient way to manage roles, because the group owner doesn't need to be an admin or app owner.

Advantages of this approach:

- Simple programming model.
- Roles are specific to the application. The role claims for one application are not sent to another application.
- If the customer removes the application from their Azure AD tenant, the roles go away.
- The application doesn't need any extra Azure AD permissions, other than reading the user's profile.

Drawbacks:

- Customers without Azure AD Premium cannot assign app roles to security groups. For these customers, all app role assignments to users must be done individually, by an administrator or an owner of the app.
- If you have a backend web API which is separate from the web app, the app role assignments for the web app don't apply to the web API. For more discussion of this point, see [Securing a backend web API](#).

Implementation

Define the roles. The SaaS provider declares the app roles in the [application manifest](#). For example, here is the manifest entry for the Surveys app:

```

"appRoles": [
  {
    "allowedMemberTypes": [
      "User"
    ],
    "description": "Creators can create Surveys",
    "displayName": "SurveyCreator",
    "id": "1b4f816e-5eaf-48b9-8613-7923830595ad",
    "isEnabled": true,
    "value": "SurveyCreator"
  },
  {
    "allowedMemberTypes": [
      "User"
    ],
    "description": "Administrators can manage the Surveys in their tenant",
    "displayName": "SurveyAdmin",
    "id": "c20e145e-5459-4a6c-a074-b942bbd4cf1",
    "isEnabled": true,
    "value": "SurveyAdmin"
  }
],

```

The `value` property appears in the role claim. The `id` property is the unique identifier for the defined role. Always generate a new GUID value for `id`.

Assign users. When a new customer signs up, the application is registered in the customer's Azure AD tenant. At this point, an Azure AD admin for that tenant or an app owner (under Enterprise apps) can assign app roles to users.

NOTE

As noted earlier, customers with Azure AD Premium can also assign app roles to security groups.

The following screenshot from the Azure portal shows users and groups for the Survey application. Admin and Creator are groups, assigned the SurveyAdmin and SurveyCreator app roles, respectively. Alice is a user who was assigned the SurveyAdmin app role directly. Bob and Charles are users that have not been directly assigned an app role.

DISPLAY NAME	OBJECT TYPE	ROLE ASSIGNED
Admin	Group	SurveyAdmin
Alice	User	SurveyAdmin
Bob	User	Default Access
Charles	User	Default Access
Creators	Group	SurveyCreator

As shown in the following screenshot, Charles is part of the Admin group, so he inherits the SurveyAdmin role. In the case of Bob, he has not been assigned an app role yet.

The screenshot shows the Azure portal interface for managing group members. On the left, there's a sidebar with 'Overview', 'Properties', and 'Members'. The 'Members' option is selected and highlighted in blue. The main area shows a table with columns 'NAME' and 'TYPE'. One row is visible, showing 'Charles' with a user icon and 'User' under 'TYPE'. There's also a button labeled '+ Add members'.

NOTE

An alternative approach is for the application to assign app roles programmatically, using the Azure AD Graph API. However, this requires the application to obtain write permissions for the customer's Azure AD directory, which is a high privilege that is usually unnecessary.

Get role claims. When a user signs in, the application receives the user's assigned role(s) in a claim with type `http://schemas.microsoft.com/ws/2008/06/identity/claims/role` (the `roles` claim in a JWT token).

A user can be assigned multiple roles, or no role. In your authorization code, don't assume the user has exactly one role claim. Instead, write code that checks whether a particular claim value is present:

```
if (context.User.HasClaim(ClaimTypes.Role, "Admin")) { ... }
```

Roles using Azure AD security groups

In this approach, roles are represented as Azure AD security groups. The application assigns permissions to users based on their security group memberships.

Advantages:

- For customers who do not have Azure AD Premium, this approach enables the customer to use security groups to manage role assignments.

Disadvantages:

- Complexity. Because every tenant sends different group claims, the app must keep track of which security groups correspond to which application roles, for each tenant.
- As users belong to more groups, access tokens grow to include more claims. After a certain limit, Azure AD includes an "overage" claim to limit the token size; see [Microsoft identity platform access tokens](#). Application roles avoid this issue because they are scoped to the specific application.

Implementation

In the application manifest, set the `groupMembershipClaims` property to "SecurityGroup". This is needed to get group membership claims from Azure AD.

```
{
  // ...
  "groupMembershipClaims": "SecurityGroup",
}
```

When a new customer signs up, the application instructs the customer to create security groups for the roles

needed by the application. The customer then needs to enter the group object IDs into the application. The application stores these in a table that maps group IDs to application roles, per tenant.

NOTE

Alternatively, the application could create the groups programmatically, using the Microsoft Graph API. This could be less error prone, but requires the application to obtain privileged read/write permissions for the customer's directory. Many customers might be unwilling to grant this level of access.

When a user signs in:

1. The application receives the user's groups as claims. The value of each claim is the object ID of a group.
2. Azure AD limits the number of groups sent in the token. If the number of groups exceeds this limit, Azure AD sends a special "overage" claim. If that claim is present, the application must query the Azure AD Graph API to get all of the groups to which that user belongs. For details, see [Groups overage claim](#).
3. The application looks up the object IDs in its own database, to find the corresponding application roles to assign to the user.
4. The application adds a custom claim value to the user principal that expresses the application role. For example: `survey_role` = "SurveyAdmin".

Authorization policies should use the custom role claim, not the group claim.

Roles using an application role manager

With this approach, application roles are not stored in Azure AD at all. Instead, the application stores the role assignments for each user in its own DB — for example, using the **RoleManager** class in ASP.NET Identity.

Advantages:

- The app has full control over the roles and user assignments.

Drawbacks:

- More complex, harder to maintain.
- Cannot use Azure AD security groups to manage role assignments.
- Stores user information in the application database, where it can get out of sync with the tenant's Azure AD directory, as users are added or removed.

[Next](#)

Role-based and resource-based authorization

3/10/2022 • 5 minutes to read • [Edit Online](#)



Sample code

Our [reference implementation](#) is an ASP.NET Core application. In this article we'll look at two general approaches to authorization, using the authorization APIs provided in ASP.NET Core.

- **Role-based authorization.** Authorizing an action based on the roles assigned to a user. For example, some actions require an administrator role.
- **Resource-based authorization.** Authorizing an action based on a particular resource. For example, every resource has an owner. The owner can delete the resource; other users cannot.

A typical app will employ a mix of both. For example, to delete a resource, the user must be the resource owner *or* an admin.

Role-based authorization

The [Tailspin Surveys](#) application defines the following roles:

- Administrator. Can perform all CRUD operations on any survey that belongs to that tenant.
- Creator. Can create new surveys
- Reader. Can read any surveys that belong to that tenant

Roles apply to *users* of the application. In the Surveys application, a user is either an administrator, creator, or reader.

For a discussion of how to define and manage roles, see [Application roles](#).

Regardless of how you manage the roles, your authorization code will look similar. ASP.NET Core has an abstraction called [authorization policies](#). With this feature, you define authorization policies in code, and then apply those policies to controller actions. The policy is decoupled from the controller.

Create policies

To define a policy, first create a class that implements `IAuthorizationRequirement`. It's easiest to derive from `AuthorizationHandler`. In the `Handle` method, examine the relevant claim(s).

Here is an example from the Tailspin Surveys application:

```
public class SurveyCreatorRequirement : AuthorizationHandler<SurveyCreatorRequirement>,  
IAuthorizationRequirement  
{  
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,  
SurveyCreatorRequirement requirement)  
    {  
        if (context.User.HasClaim(ClaimTypes.Role, Roles.SurveyAdmin) ||  
            context.User.HasClaim(ClaimTypes.Role, Roles.SurveyCreator))  
        {  
            context.Succeed(requirement);  
        }  
        return Task.FromResult(0);  
    }  
}
```

This class defines the requirement for a user to create a new survey. The user must be in the SurveyAdmin or SurveyCreator role.

In your startup class, define a named policy that includes one or more requirements. If there are multiple requirements, the user must meet *every* requirement to be authorized. The following code defines two policies:

```
services.AddAuthorization(options =>
{
    options.AddPolicy(PolicyNames.RequireSurveyCreator,
        policy =>
    {
        policy.AddRequirements(new SurveyCreatorRequirement());
        policy.RequireAuthenticatedUser(); // Adds DenyAnonymousAuthorizationRequirement
        // By adding the CookieAuthenticationDefaults.AuthenticationScheme, if an authenticated
        // user is not in the appropriate role, they will be redirected to a "forbidden" page.
        policy.AddAuthenticationSchemes(CookieAuthenticationDefaults.AuthenticationScheme);
    });

    options.AddPolicy(PolicyNames.RequireSurveyAdmin,
        policy =>
    {
        policy.AddRequirements(new SurveyAdminRequirement());
        policy.RequireAuthenticatedUser();
        policy.AddAuthenticationSchemes(CookieAuthenticationDefaults.AuthenticationScheme);
    });
});
```

This code also sets the authentication scheme, which tells ASP.NET which authentication middleware should run if authorization fails. In this case, we specify the cookie authentication middleware, because the cookie authentication middleware can redirect the user to a "Forbidden" page. The location of the Forbidden page is set in the `AccessDeniedPath` option for the cookie middleware; see [Configuring the authentication middleware](#).

Authorize controller actions

Finally, to authorize an action in an MVC controller, set the policy in the `[Authorize]` attribute:

```
[Authorize(Policy = PolicyNames.RequireSurveyCreator)]
public IActionResult Create()
{
    var survey = new SurveyDTO();
    return View(survey);
}
```

In earlier versions of ASP.NET, you would set the `Roles` property on the attribute:

```
// old way
[Authorize(Roles = "SurveyCreator")]
```

This is still supported in ASP.NET Core, but it has some drawbacks compared with authorization policies:

- It assumes a particular claim type. Policies can check for any claim type. Roles are just a type of claim.
- The role name is hard-coded into the attribute. With policies, the authorization logic is all in one place, making it easier to update or even load from configuration settings.
- Policies enable more complex authorization decisions (for example, $age \geq 21$) that can't be expressed by simple role membership.

Resource-based authorization

Resource based authorization occurs whenever the authorization depends on a specific resource that will be

affected by an operation. In the Tailspin Surveys application, every survey has an owner and zero-to-many contributors.

- The owner can read, update, delete, publish, and unpublish the survey.
- The owner can assign contributors to the survey.
- Contributors can read and update the survey.

Note that "owner" and "contributor" are not application roles; they are stored per survey, in the application database. To check whether a user can delete a survey, for example, the app checks whether the user is the owner for that survey.

In ASP.NET Core, implement resource-based authorization by deriving from **AuthorizationHandler** and overriding the **Handle** method.

```
public class SurveyAuthorizationHandler : AuthorizationHandler<OperationAuthorizationRequirement, Survey>
{
    protected override void HandleRequirementAsync(AuthorizationHandlerContext context,
OperationAuthorizationRequirement operation, Survey resource)
    {
    }
}
```

Notice that this class is strongly typed for Survey objects. Register the class for DI on startup:

```
services.AddSingleton<IAuthorizationHandler>(factory =>
{
    return new SurveyAuthorizationHandler();
});
```

To perform authorization checks, use the **IAuthorizationService** interface, which you can inject into your controllers. The following code checks whether a user can read a survey:

```
if (await _authorizationService.AuthorizeAsync(User, survey, Operations.Read) == false)
{
    return StatusCode(403);
}
```

Because we pass in a `Survey` object, this call will invoke the `SurveyAuthorizationHandler`.

In your authorization code, a good approach is to aggregate all of the user's role-based and resource-based permissions, then check the aggregate set against the desired operation. Here is an example from the Surveys app. The application defines several permission types:

- Admin
- Contributor
- Creator
- Owner
- Reader

The application also defines a set of possible operations on surveys:

- Create
- Read
- Update
- Delete

- Publish
- Unpublish

The following code creates a list of permissions for a particular user and survey. Notice that this code looks at both the user's app roles, and the owner/contributor fields in the survey.

```
public class SurveyAuthorizationHandler : AuthorizationHandler<OperationAuthorizationRequirement, Survey>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
OperationAuthorizationRequirement requirement, Survey resource)
    {
        var permissions = new List<UserPermissionType>();
        int surveyTenantId = context.User.GetSurveyTenantIdValue();
        int userId = context.User.GetSurveyUserIdValue();
        string user = context.User.GetUserName();

        if (resource.TenantId == surveyTenantId)
        {
            // Admin can do anything, as long as the resource belongs to the admin's tenant.
            if (context.User.HasClaim(ClaimTypes.Role, Roles.SurveyAdmin))
            {
                context.Succeed(requirement);
                return Task.FromResult(0);
            }

            if (context.User.HasClaim(ClaimTypes.Role, Roles.SurveyCreator))
            {
                permissions.Add(UserPermissionType.Creator);
            }
            else
            {
                permissions.Add(UserPermissionType.Reader);
            }

            if (resource.OwnerId == userId)
            {
                permissions.Add(UserPermissionType.Owner);
            }
        }
        if (resource.Contributors != null && resource.Contributors.Any(x => x.UserId == userId))
        {
            permissions.Add(UserPermissionType.Contributor);
        }

        if (ValidateUserPermissions[requirement](permissions))
        {
            context.Succeed(requirement);
        }
        return Task.FromResult(0);
    }
}
```

In a multitenant application, you must ensure that permissions don't "leak" to another tenant's data. In the Surveys app, the Contributor permission is allowed across tenants—you can assign someone from another tenant as a contributor. The other permission types are restricted to resources that belong to that user's tenant. To enforce this requirement, the code checks the tenant ID before granting the permission. (The `TenantId` field as assigned when the survey is created.)

The next step is to check the operation (such as read, update, or delete) against the permissions. The Surveys app implements this step by using a lookup table of functions:

```
static readonly Dictionary<OperationAuthorizationRequirement, Func<List<UserPermissionType>, bool>>
ValidateUserPermissions
    = new Dictionary<OperationAuthorizationRequirement, Func<List<UserPermissionType>, bool>>

    {
        { Operations.Create, x => x.Contains(UserPermissionType.Creator) },

        { Operations.Read, x => x.Contains(UserPermissionType.Creator) ||
            x.Contains(UserPermissionType.Reader) ||
            x.Contains(UserPermissionType.Contributor) ||
            x.Contains(UserPermissionType.Owner) },

        { Operations.Update, x => x.Contains(UserPermissionType.Contributor) ||
            x.Contains(UserPermissionType.Owner) },

        { Operations.Delete, x => x.Contains(UserPermissionType.Owner) },

        { Operations.Publish, x => x.Contains(UserPermissionType.Owner) },

        { Operations.UnPublish, x => x.Contains(UserPermissionType.Owner) }
    };
}
```

[Next](#)

Cache access tokens

3/10/2022 • 2 minutes to read • [Edit Online](#)



Sample code

It's relatively expensive to get an OAuth access token, because it requires an HTTP request to the token endpoint. Therefore, it's good to cache tokens whenever possible. The [Microsoft Authentication Library for .NET \(MSAL.NET\)](#) (MSAL) caches tokens obtained from Azure AD, including refresh tokens.

Some implementations include MSAL are in-memory cache and distributed cache. This option is set in the ConfigureServices method of the Startup class of the web application. To acquire a token for the downstream API, you'll need to `.EnableTokenAcquisitionToCallDownstreamApi()`.

The Surveys app uses distributed token cache that stores data in the backing store. The app uses a Redis cache as the backing store. Every server instance in a server farm reads/writes to the same cache, and this approach scales to many users.

For a single-instance web server, you could use the ASP.NET Core [in-memory cache](#). (This is also a good option for running the app locally during development.)

```
services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApp(
        options =>
    {
        Configuration.Bind("AzureAd", options);
        options.Events = new SurveyAuthenticationEvents(loggerFactory);
        options.SignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        options.Events.OnTokenValidated += options.Events.TokenValidated;
    })
    .EnableTokenAcquisitionToCallDownstreamApi()
    .AddDownstreamWebApi(configOptions.SurveyApi.Name, Configuration.GetSection("SurveyApi"))
    .AddDistributedTokenCaches();

    services.AddStackExchangeRedisCache(options =>
    {
        options.Configuration = configOptions.Redis.Configuration;
        options.InstanceName = "TokenCache";
    });
}
```

The configuration for SurveyApi is specified in appsettings.json.

```
"SurveyApi": {
    "BaseUrl": "https://localhost:44301",
    "Scopes": "https://test.onmicrosoft.com/surveys.webapi/surveys.access",
    "Name": "SurveyApi"
},
```

Encrypting cached tokens

Tokens are sensitive data, because they grant access to a user's resources. (Moreover, unlike a user's password, you can't just store a hash of the token.) Therefore, it's critical to protect tokens from being compromised.

The Redis-backed cache is protected by a password, but if someone obtains the password, they could get all of

the cached access tokens. The MSAL token cache is encrypted.

Acquire the token

The Survey application calls the downstream web API from the page constructor.

```
public class SurveyService : ISurveyService
{
    private string _serviceName;
    private readonly IDownstreamWebApi _downstreamWebApi;

    public SurveyService(HttpClientService factory, IDownstreamWebApi downstreamWebApi,
IOptions<ConfigurationOptions> configOptions)
    {
        _serviceName = configOptions.Value.SurveyApi.Name;
        _downstreamWebApi = downstreamWebApi;
    }

    public async Task<SurveyDTO> GetSurveyAsync(int id)
    {
        return await _downstreamWebApi.CallWebApiForUserAsync<SurveyDTO>(_serviceName,
            options =>
        {
            options.HttpMethod = HttpMethod.Get;
            options.RelativePath = $"surveys/{id}";
        });
    }
}
```

Another way is to inject an `ITokenAcquisition` service in the controller. For more information, see [Acquire and cache tokens using the Microsoft Authentication Library \(MSAL\)](#)

[Next](#)

Next steps

- [Token cache serialization in MSAL.NET](#)
- [Acquire and cache tokens using the Microsoft Authentication Library \(MSAL\)](#)

Related resources

- [Identity management in multitenant applications](#)
- [Secure a backend web API for multitenant applications](#)

Getting started with Azure IoT solutions

3/10/2022 • 5 minutes to read • [Edit Online](#)

IoT (Internet of Things) is a collection of managed and platform services that connect and control IoT assets. For example, consider an industrial motor connected to the cloud. The motor collects and sends temperature data. This data is used to evaluate whether the motor is performing as expected. This information can then be used to prioritize a maintenance schedule for the motor.

Azure IoT supports a large range of devices, including industrial equipment, microcontrollers, sensors, and many others. When connected to the cloud, these devices can send data to your IoT solution. You can use the solution to monitor and manage the devices to achieve your objectives more efficiently.

This document links to guides that you can use to accelerate your creation of IoT solutions.

Industry specific IoT reference architectures

IoT solutions solve specific business challenges in unique industries. In this section, we provide reference designs for industry-specific scenarios.

Manufacturing and industrial

- [Connected factory hierarchy service](#): A hierarchy service enables your business to centrally define how production assets are organized within factories.
- [Connected factory signal pipeline](#): It can be problematic to interconnect the heterogenous legacy and modern devices that often coexist in a factory. The connected factory signal pipeline solves this problem by introducing a common configuration interface to connect heterogenous devices.
- [Predictive maintenance with IoT Edge](#): The Internet-of-things (IoT) Edge brings data processing and storage close to the data source, enabling fast, consistent responses with reduced dependency on cloud connectivity and resources. Edge computing can incorporate artificial intelligence (AI) and machine learning (ML) models to create intelligent edge devices and networks that can integrate with the cloud.
- [Condition monitoring for industrial IoT](#): OPC UA (Open Platform Communication Unified Architecture) can be used to monitor equipment parameters to discover anomalies before they become critical issues.
- [End-to-end manufacturing using computer vision on the edge](#): Fully automated smart factories use artificial intelligence (AI) and machine learning (ML) to analyze data, run systems, and improve processes over time. In manufacturing, computer vision on the edge is an increasingly popular Internet of Things (IoT) application used in safety and quality assurance applications.

Smart buildings

- [Create smart places by using Azure Digital Twins](#): Smart places are physical environments that bring together connected devices and data sources. They can include buildings, college campuses, corporate campuses, stadiums, and cities. Smart places provide value by helping property owners, facility managers, and occupants operate and maintain sites.

Retail

- [Buy online, pickup in store \(BOPIS\)](#): During the COVID-19 pandemic, many customers made fewer trips to the market and many preferred to buy their merchandise online and pick it up in the store (BOPIS) which is also known as curbside pickup.

Automotive

- [Process real-time vehicle data using IoT](#): Vehicle data ingestion, processing, and visualization are key capabilities needed to create connected car solutions. By capturing and analyzing this data, we can decipher valuable insights and create new solutions.

Sustainability

- [Project 15 Open Platform IoT sustainability](#): The mission of Project 15 from Microsoft is to empower scientists and conservationists around the world.

Cross industry

- [Automated guided vehicles fleet control](#): Automotive manufacturing relies on automated guided vehicles (AGVs) to deliver parts to assembly lines. AGVs are a mission-critical part of just-in-time manufacturing and automated shop-floor logistics.
- [Environment monitoring and supply chain optimization with IoT](#): Environmental monitoring has become an important activity in the global supply chain. It provides key signals that help drive real-time decisions that can impact suppliers and logistics.
- [Real-time asset tracking and management](#): You can use Azure IoT Central and other Azure services to track and manage assets in real time.

IoT architecture patterns and guides

This section shares guides and building blocks that are useful when building IoT solutions. Patterns address key areas of the IoT solution that you should consider. IoT architecture patterns are reusable building blocks that you can leverage to create IoT solutions. They are generic and can be used across different industry verticals.

Patterns

- [Real-time IoT updates](#): Internet of Things (IoT) applications often need real-time data from IoT devices. Unlike traditional polling apps in which a client asks a device for state changes, clients can receive updates from devices in real time.
- [Scale IoT solutions with deployment stamps](#): The deployment stamping strategy in an Internet-of-Things (IoT) solution supports scaling up the number of connected IoT devices by replicating stamps. Stamps are discrete units of core solution components that optimally support a defined number of devices.
- [Azure IoT client SDK support for third-party token servers](#): Learn how Azure IoT Hub supports shared access signature (SAS) token authentication in the client SDKs.
- [Efficient Docker image deployment](#): Edge devices are typically provisioned by deploying software container images. You can use a reliable and resilient deployment capability for situations that have limited, intermittent, or low bandwidth.
- [IoT analytics with Azure Data Explorer](#): Achieve near real-time analytics over fast flowing, high volume streaming data from IoT devices.
- [IoT Edge data storage and processing](#): An Internet of Things (IoT) solution might require that an on-premises edge network provide computing and data collecting, rather than the cloud. Edge devices often meet these needs better than the cloud.

Guides for building IoT solutions

- [Compare IoT solution approaches](#): There are many options for building and deploying your IoT cloud solutions. The technologies and services you choose depends on your scenario's development, deployment, and management needs. Use the linked article to compare PaaS and aPaaS approaches.
- [Vision with Azure IoT Edge](#): Artificial intelligence for image analytics spans a wide variety of industries including manufacturing, retail, healthcare, and the public sector.

- [Move an IoT solution from test to production](#) provides a list of best practices you should follow when you move from test to production.
- [Azure industrial IoT analytics guidance](#): You can build Industrial IoT (IIoT) analytics solutions using Azure PaaS (Platform as a Service) components. Such an IIoT analytics solution relies on real-time and historical data from industrial devices and control systems.

Next steps

Learn about the different Azure IoT services:

- [Azure IoT documentation](#)
- [Azure IoT Central documentation](#)
- [Azure IoT Hub](#)
- [Azure IoT Hub Device Provisioning Service](#)
- [Azure IoT Edge documentation](#)

Related resources

See the related IoT architecture guides:

- [Azure IoT reference architecture](#)
- [IoT solutions conceptual overview](#)
- [Choose an Internet of Things \(IoT\) solution in Azure](#)

See the related IoT solution guides for COVID-19:

- [COVID-19 safe environments with IoT Edge monitoring and alerting](#)
- [IoT connected light, power, and internet for emerging markets](#)
- [UVEN smart and secure disinfection and lighting](#)
- [Cognizant Safe Buildings with IoT and Azure](#)

Vision AI solutions with Azure IoT Edge

3/10/2022 • 3 minutes to read • [Edit Online](#)

This series of articles describes how to plan and design a computer vision workload that uses [Azure IoT Edge](#). You can run Azure IoT Edge on devices, and integrate with Azure Machine Learning, Azure Storage, Azure App Services, and Power BI for end-to-end vision AI solutions.

Visually inspecting products, resources, and environments is critical for many endeavors. Human visual inspection and analytics are subject to inefficiency and inaccuracy. Enterprises now use deep learning artificial neural networks called *convolutional neural networks* (CNNs) to emulate human vision. Using CNNs for automated image input and analysis is commonly called *computer vision* or *vision AI*.

Technologies like containerization support portability, which allows migrating vision AI models to the network edge. You can train vision inference models in the cloud, containerize the models, and use them to create custom modules for Azure IoT Edge runtime-enabled devices. Deploying vision AI solutions at the edge yields performance and cost benefits.

Use cases

Use cases for vision AI span manufacturing, retail, healthcare, and the public sector. Typical vision AI use cases include quality assurance, safety, and security.

Quality assurance

In manufacturing environments, vision AI can inspect parts and processes fast and accurately. Automated quality inspection can:

- Monitor manufacturing process consistency.
- Check proper product assembly.
- Provide early defect notifications.

For an example scenario for this use case, see [User scenario 1: Quality control](#).

Safety and security

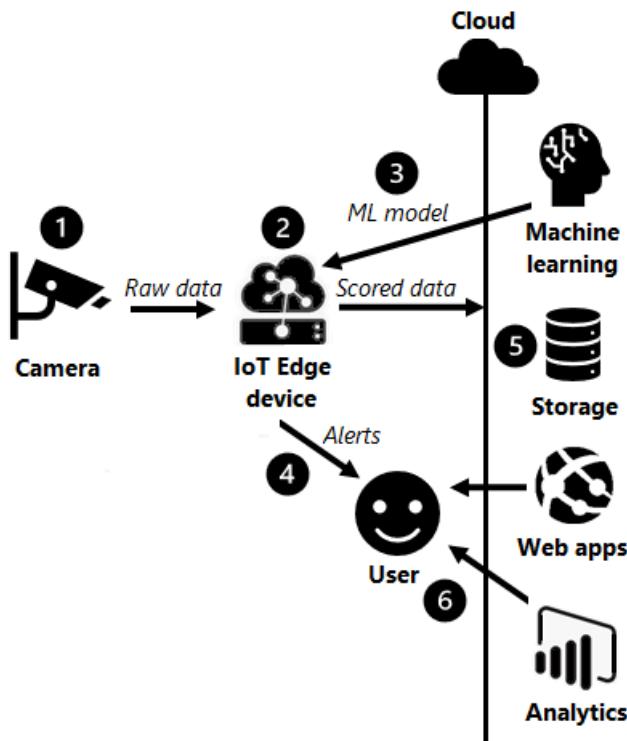
Automated visual monitoring can scan for potential safety and security issues. Automation can provide more time to respond to incidents, and more opportunities to reduce risk. Automated safety monitoring can:

- Track compliance with personal protective equipment guidelines.
- Monitor and alert on entry into unauthorized zones.
- Alert on unidentified objects.
- Record unreported close calls or pedestrian-equipment near-misses.

For an example scenario for this use case, see [User scenario 2: Safety](#).

Architecture

Vision AI solutions for IoT Edge involve several components and processes. The articles in this series provide in-depth planning and design guidance for each area.



1. Cameras capture the image data for input into the IoT Edge vision AI system. See [Camera selection for Azure IoT Edge vision AI](#).
2. Hardware acceleration on IoT Edge devices provides the necessary processing power for computer graphics and AI algorithms. See [Hardware acceleration in Azure IoT Edge vision AI](#).
3. ML models deployed as IoT Edge modules score the incoming image data. See [Machine learning in Azure IoT Edge vision AI](#).
4. Image scores that need attention trigger automatic alerts. See [Alert persistence in Azure IoT Edge vision AI](#).
5. The IoT Edge device sends relevant image data and metadata to the cloud for storage. Stored data is used for ML retraining, troubleshooting, and analytics. See [Image storage and management for Azure IoT Edge vision AI](#).
6. Users interact with the system through user interfaces like apps, visualizations, and dashboards. See [User interfaces and scenarios in Azure IoT Edge vision AI](#).

Considerations

Reasons to migrate computer vision workloads from the cloud to the edge include performance and cost.

Performance considerations

- Exporting less data to the cloud relieves strain on network infrastructure that can cause performance issues.
- Scoring data locally helps prevent unacceptable response latency.
- Local alerting avoids delay and added complexity.

For example, a person entering an unauthorized area might need immediate intervention. Positioning the scoring model near the data ingestion point allows near real-time image scoring and alerting.

Cost considerations

Scoring data locally and sending only relevant data to the cloud can improve the return on investment (ROI) of a computer vision initiative. IoT Edge custom vision modules can score image data per ML models, and send only images deemed relevant with reasonable confidence to the cloud for further processing. Sending only selected images reduces the amount of data going to the cloud and lowers costs.

Next steps

To continue with this series about IoT Edge vision AI, go on to the next article:

[Camera selection for Azure IoT Edge vision AI](#)

To learn more about CNNs, vision AI, Azure Machine Learning, and Azure IoT Edge, see the following documentation:

- [Azure IoT Edge documentation](#)
- [Azure Machine Learning documentation](#)
- [Tutorial: Perform image classification at the edge with Custom Vision Service](#)
- [What is Computer Vision?](#)
- [What is Azure Video Analyzer? \(preview\)](#)
- [Azure Kinect DK developer kit documentation](#)
- [Open Neural Network Exchange \(ONNX\) ML framework](#)
- [Model management deep neural network \(MMdnn\) tool](#)

Related resources

For more computer vision architectures, examples, and ideas that use Azure IoT, see the following articles:

- [Getting started with Azure IoT solutions](#)
- [End-to-end manufacturing using computer vision at the edge](#)
- [Video capture and analytics for retail](#)
- [Connected factory hierarchy service](#)
- [Connected factory signal pipeline](#)
- [Create smart places by using Azure Digital Twins](#)
- [Deploy AI and ML computing on-premises and to the edge](#)

Camera selection for Azure IoT Edge vision AI

3/10/2022 • 7 minutes to read • [Edit Online](#)

One of the most critical components in a computer vision system is the camera. The camera must capture and present images that artificial intelligence (AI) or machine learning (ML) models can evaluate and identify correctly. This article provides an in-depth understanding of different camera types, capabilities, and considerations.

Types of cameras

Camera types include area scan, line scan, and embedded smart cameras. There are many different manufacturers for these cameras. Select a vendor that fits your specific needs.

Area scan cameras

Area scan cameras generate a traditional camera image. This camera typically has a matrix of pixel sensors. The camera captures a 2D image and sends it to the Azure IoT Edge hardware for evaluation.

Area scan cameras look at a large area, and are good for detecting changes. Examples of workloads that can use area scan cameras are workplace safety, or detecting or counting objects in an environment.

Line scan cameras

A line scan camera has a single row of linear pixel sensors. The camera takes 1-pixel-width images in quick succession, stitches them together into a video stream, and sends the stream to the IoT Edge device for processing.

Line scan cameras are good for vision workloads where items are either moving past the camera, or need to be rotated to detect defects. The line scan camera then produces a continuous image stream for evaluation.

Examples of workloads that work best with line scan cameras are:

- Item defect detection on parts that are moving on a conveyor belt
- Workloads that require spinning to see a cylindrical object
- Workloads that require rotation

Embedded smart cameras

An embedded smart camera is a self-contained, standalone system that can process as well as acquire images. Embedded smart cameras can use either an area scan or a line scan camera for capturing images, although a line scan smart camera is rare. These cameras usually have either an RS232 or an Ethernet output port, so they can integrate directly into a programmable logic controller (PLC) or other industrial IoT (IIoT) controller.

Camera features

There are several features to consider when selecting a camera for a vision workload. The following sections discuss sensor size, resolution, and speed. Other camera features to consider include:

- Lens selection
- Focal length
- Monochrome or color depth
- Stereo depth
- Triggers
- Physical size

- Support

Camera manufacturers can help you understand what specific features your application requires.

Sensor size

Sensor size is one of the most important factors to evaluate in a camera. The sensor is the hardware within a camera that captures the target and converts it into signals, which then produce an image. The sensor contains millions of semiconducting photodetectors called *photosites*.

A higher megapixel count doesn't always result in a better image. For example, a camera with 12 million photosites and a 1-inch sensor produces a clearer, sharper image than a camera with 12 million photosites and a $\frac{1}{2}$ -inch sensor. Cameras for computer vision workloads usually have sensor sizes between $\frac{1}{4}$ inch and 1 inch. Some cases might require much larger sensors.

Choose larger sensors if your vision workload has:

- A need for precision measurements
- Lower light conditions
- Shorter exposure times or fast-moving items

Resolution

Resolution is another important factor in camera selection. You need higher resolution cameras if your workload:

- Must identify fine features, such as the writing on an integrated circuit chip
- Is trying to detect faces
- Needs to identify vehicles from a distance

The following images show the problem with using the wrong resolution for a given use case. Both images were taken 20 feet away from the car. The small red boxes represent one pixel.

- The following image was taken with 480 horizontal pixels:



- The following image was taken with 5184 horizontal pixels:



Speed

If your vision workload requires capturing many images per second, two factors are important. The first factor is the speed of the camera interface connection. The second factor is the type of sensor. Sensors come in two types, [charge-coupled devices \(CCD\)](#) and [active-pixel sensors \(CMOS\)](#). CMOS sensors have a direct readout from the photosites, so they typically offer a higher frame rate.

Camera placement

The items you need to capture in your vision workload determine the locations and angles for camera placement. Camera location can also interact with sensor type, lens type, and camera body type. Two of the most critical factors for determining camera placement are lighting and field of view.

Camera lighting

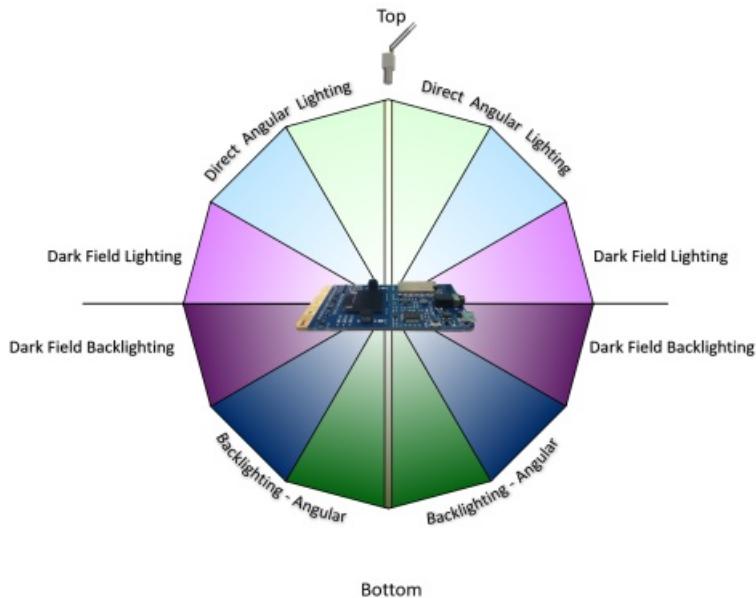
In a computer vision workload, lighting is critical to camera placement. You can apply several different lighting conditions. Lighting conditions that are useful for one vision workload might produce undesirable effects in a different workload.

There are several common lighting types for computer vision workloads:

- *Direct lighting* is the most common lighting condition. The light source is projected at the object to be captured.
- *Line lighting* is a single array of lights most used with line scan cameras. Line lighting creates a single line of light at the focus of the camera.
- *Diffused lighting* illuminates an object but prevents harsh shadows. Diffused lighting is mostly used around specular, or reflective, objects.
- *Axial diffused lighting* is often used with highly reflective objects, or to prevent shadows on the part to capture.
- *Back lighting* is used behind the object, producing a silhouette of the object. Back lighting is most useful for measurements, edge detection, or object orientation.
- *Custom grid lighting* is a structured lighting condition that lays out a grid of light on the object. The known grid projection provides more accurate measurements of item components, parts, and placement.
- *Strobe lighting* is used for high speed moving parts. The strobe must be in sync with the camera to take a freeze of the object for evaluation. Strobe lighting helps prevent motion blurring effects.

- *Dark field lighting* uses several lights with different angles to the part to be captured. For example, if the part is lying flat on a conveyor belt, the lights are at a 45-degree angle to the belt. Dark field lighting is most useful with highly reflective clear objects, and is commonly used for *lens scratch detections*.

The following figure demonstrates the angular placement of light sources:



Field of view

In planning a vision workload, you need to know about the field of view (FOV) of the objects you're evaluating. FOV plays a part in camera selection, sensor selection, and lens configuration. FOV components include:

- Distance to objects. For example, is the object being monitored on a conveyor belt with the camera two feet above it, or across a parking lot? Camera sensors and lens configurations are different for different distances.
- Area of coverage. Is the area that the computer vision is trying to monitor small or large? This factor directly correlates to the camera's resolution, lens, and sensor type.
- Direction of the sun. If the computer vision workload is outdoors, you should consider the direction of the sun throughout the day. The angle of the sun as it moves might impact the computer vision model. If the camera gets direct sunlight in the lens, it might be blinded until the angle of the sun changes. If the sun casts a shadow over the object being monitored, the object might be obscured.
- Camera angle to the objects. If the camera is too high or too low, it might miss the details that the workload is trying to capture.

Communication interface

In planning a computer vision workload, it's important to understand how the camera output interacts with the rest of the system. There are several standard ways that cameras communicate to IoT Edge devices:

- [Real Time Streaming Protocol \(RTSP\)](#) is an application-level network protocol that controls streaming video servers. RTSP transfers real-time video data from the camera to the IoT Edge compute endpoint over a TCP/IP connection.
- [Open Network Video Interface Forum \(ONVIF\)](#) is a global, open industry forum that develops open standards for IP-based cameras. These standards describe communication between IP cameras and downstream systems, interoperability, and open source.
- [Universal Serial Bus \(USB\)](#)-connected cameras connect over the USB port directly to the IoT Edge compute device. This connection is less complex, but limits the distance the camera can be located from the IoT Edge device.
- [Camera Serial Interface \(CSI\)](#) includes several standards from the Mobile Industry Processor Interface

(MIPI) Alliance. CSI describes how to communicate between a camera and a host processor. CSI-2, released in 2005, has several layers:

- Physical layer (either C-PHY or D-PHY)
- Lane merger layer
- Low-level protocol layer
- Pixel to byte conversion layer
- Application layer

[CSI-2 v3.0](#) added support for RAW-24 color depth, Unified Serial Link, and Smart Region of Interest.

Next steps

[Hardware acceleration in Azure IoT Edge vision AI](#)

Hardware acceleration for Azure IoT Edge vision AI

3/10/2022 • 2 minutes to read • [Edit Online](#)

Computer graphics and artificial intelligence (AI) require large amounts of computing power. A critical factor in designing Azure IoT Edge vision AI projects is the degree of hardware acceleration the solution needs.

Hardware accelerators such as graphics processing units (GPUs), field programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs) are cost effective ways to improve performance.

Computing hardware types

The following sections describe the main types of computing hardware for IoT Edge vision components.

CPU

A central processing unit (CPU) is the default option for most general purpose computing. A CPU might be sufficient for vision workloads where timing isn't critical. However, workloads that involve critical timing, multiple camera streams, or high frame rates need specific hardware acceleration.

GPU

A GPU is the default processor for high-end computer graphics cards. High performance computer (HPC) scenarios, data mining, and AI or machine learning (ML) workloads all use GPUs. Vision workloads use GPUs' massive parallel computing power to accelerate pixel data processing. The downside to a GPU is its higher power consumption, which is a critical consideration in edge workloads.

FPGA

FPGAs are powerful, reconfigurable hardware accelerators that support the growth of deep learning neural networks. FPGA accelerators have millions of programmable gates and hundreds of I/O pins, and can do trillions of multiply accumulate (MAC) operations per second (TOPS). There are many FPGA libraries optimized for vision workloads. Some of these libraries include preconfigured interfaces to connect to downstream cameras and devices.

The usage of FPGAs in ML and IoT Edge workloads is still evolving. FPGAs tend to fall short in floating point operations, but manufacturers have made improvements in this area.

ASIC

ASICs are manufactured to do a specific task. ASICs are by far the fastest accelerators available, but are the least configurable. ASIC chips are popular because of their small size, power per watt performance, and intellectual property (IP) protection. The IP is burned into ASIC chips, making it hard to reverse engineer proprietary algorithms.

Next steps

[Machine learning and data science in Azure IoT Edge vision AI](#)

Alerts in Azure IoT Edge vision AI

3/10/2022 • 2 minutes to read • [Edit Online](#)

In an artificial intelligence (AI) context, alerts are responses to triggering events from the AI model. The events are inferencing results based on the AI model's training.

Alerts must be monitored, because they drive certain actions. Alerts are time sensitive for processing, and must be logged for audit and further analysis. Alert events are different from operational or health events that the processing pipeline or runtime raise.

In vision AI, alerting typically occurs for triggering events related to:

- Image classification
- Movement detection or direction
- Object detection or count
- Average or total object count over time

Alert persistence

Vision AI alerts should persist locally where they're raised, and pass on to the cloud for further processing and storage. Alert persistence enables quick local response, and prevents losing critical alerts due to transient network issues.

Options to achieve alert persistence and cloud syncing include:

- Use the built-in store and forward capability of the IoT Edge runtime, which automatically syncs with Azure IoT Hub after any lost connectivity.
- Persist alerts on the host file system as log files, and periodically sync the logs to blob storage in the cloud.
- Use an [Azure IoT Edge blob storage module](#) to sync the data to Azure Blob Storage in the cloud, based on configurable policies.
- Use a local database such as [Azure SQL Edge](#) for storing data on IoT Edge, and sync with Azure SQL Database by using SQL Data Sync. Another lightweight database option is [SQLite](#).

For alerts, the best option is the built-in store and forward capability of the IoT Edge runtime. This option is the most suitable because of its time sensitivity, typically small messages size, and ease of use. For more information, see [Understand extended offline capabilities for IoT Edge devices, modules, and child devices](#).

Next step

[Image storage and management in Azure IoT Edge Vision](#)

Image storage in Azure IoT Edge vision AI

3/10/2022 • 2 minutes to read • [Edit Online](#)

Image storage and management are important functions in Azure IoT Edge computer vision solutions.

Image storage requirements include:

- Fast storage to avoid pipeline bottlenecks and data loss
- Storage and labeling at the edge and in the cloud
- Easy retrieval of stored raw images for labeling
- Categorization of images for easy retrieval
- Naming and tagging to link images with inferred metadata

You can combine Blob Storage, Azure IoT Hub, and IoT Edge in several different ways to store image data. For example:

- Use an [Azure IoT Edge blob storage module](#) to automatically sync images to Azure Blob Storage via policy.
- Store images to a local host file system, and upload them to Blob Storage by using a custom module.
- Use a local database to store images, and sync them to the cloud database.

Example storage workflow

The following steps describe a typical workflow that uses an IoT Edge blob storage module.

1. The IoT Edge blob module stores raw data locally after ingestion, with time stamping and sequence numbering to uniquely identify the image files.
2. A policy set on the IoT Edge blob module automatically uploads the image data to Azure Blob Storage, with ordering.
3. To conserve space, the IoT Edge device automatically deletes the local data after a certain time span. The device also has the *retain while uploading* option set, to ensure all images sync to the cloud before deletion.
4. Local categorization or labeling uses a module that reads images into a user interface. The label data associates to the image URI, along with coordinates and category.
5. A local database stores the image metadata, and syncs to the cloud by using telemetry messages. Local storage supports easy lookup for the user interface.
6. During a scoring run, the machine learning model detects matching patterns and generates events of interest.
 - The model sends this metadata to the cloud via telemetry that refers to the image URI.
 - Optionally, the model also stores this metadata in the local database for the edge user interface.
 - The images themselves continue to store in the IoT Edge blob module and sync to Azure Blob Storage.

Next steps

[User interface and scenarios in Azure IoT Edge vision AI](#)

User interfaces and scenarios for Azure IoT Edge vision AI

3/10/2022 • 7 minutes to read • [Edit Online](#)

This final article in the Azure IoT Edge vision AI series discusses how users interact with internet of things (IoT) and artificial intelligence (AI) solutions. The article also presents two example IoT Edge vision AI scenarios.

User interfaces

Users interact with computer systems through a user interface (UI). UI requirements vary depending on overall objectives. IoT systems usually have four UI types:

- The *administrator* UI allows full access to device provisioning, device and solution configuration, and user management. These features can be part of one solution, or separate solutions.
- An *operator* UI provides access to the solution's operational components, such as device management, alert monitoring, and configuration.
- A *consumer* UI applies only to consumer-facing solutions. The UI is similar to an operator's interface, but is limited to the devices the user owns.
- An *analytics* UI is an interactive dashboard that provides telemetry visualizations and other data analyses.

Technology choices

Here are some of the services and software you can use to create user interfaces for IoT Edge vision AI systems:

- [Azure App Service](#) is a platform for developers to quickly build, deploy, and scale web and mobile apps. App Service supports frameworks like .NET, .NET Core, Node.js, Java, PHP, Ruby, or Python. Apps can be in containers or run on any supported operating system, mobile device, or IoT Edge hardware. The fully managed App Service platform meets enterprise-grade performance, security, and compliance requirements.
- [Azure SignalR Service](#) adds real-time data communications and reporting to apps, without requiring in-depth real-time communications expertise. SignalR Service integrates easily with many Azure cloud services.
- [Azure Maps](#) is a technology for IoT visualization and computer vision projects. Azure Maps lets you create location-aware web and mobile apps by using simple and secure geospatial services, APIs, and SDKs. Azure Maps has built-in location intelligence from worldwide technology partners. You can deliver seamless experiences based on geospatial data.
- [Azure Active Directory \(Azure AD\)](#) provides single sign-on and multi-factor authentication to secure your apps and user interfaces.
- [Power BI](#) is a set of analytics services, apps, and connectors that turn data into customizable, interactive visualizations and dashboards. Power BI is available as a managed service or self-hosted package, and connects to many popular database systems and data services. With [Power BI Embedded](#), you can create customer-facing reports and dashboards, and brand them as your own apps. Power BI can conserve developer resources by automating analytics monitoring, management, and deployment.

User scenario 1: Quality control

Contoso Boards produces high-quality circuit boards used in computers. Their number one product is a motherboard.

Contoso Boards saw an increase in issues with chip placement on the board. Investigation determined that the circuit boards were being placed incorrectly on the assembly line. Contoso Boards needed a way to identify and check correct circuit board placement.

The Contoso Boards data scientists were familiar with [TensorFlow](#), and wanted to continue using it as their primary ML model structure. Contoso Boards also wanted to centralize management of several assembly lines that produce the motherboards.

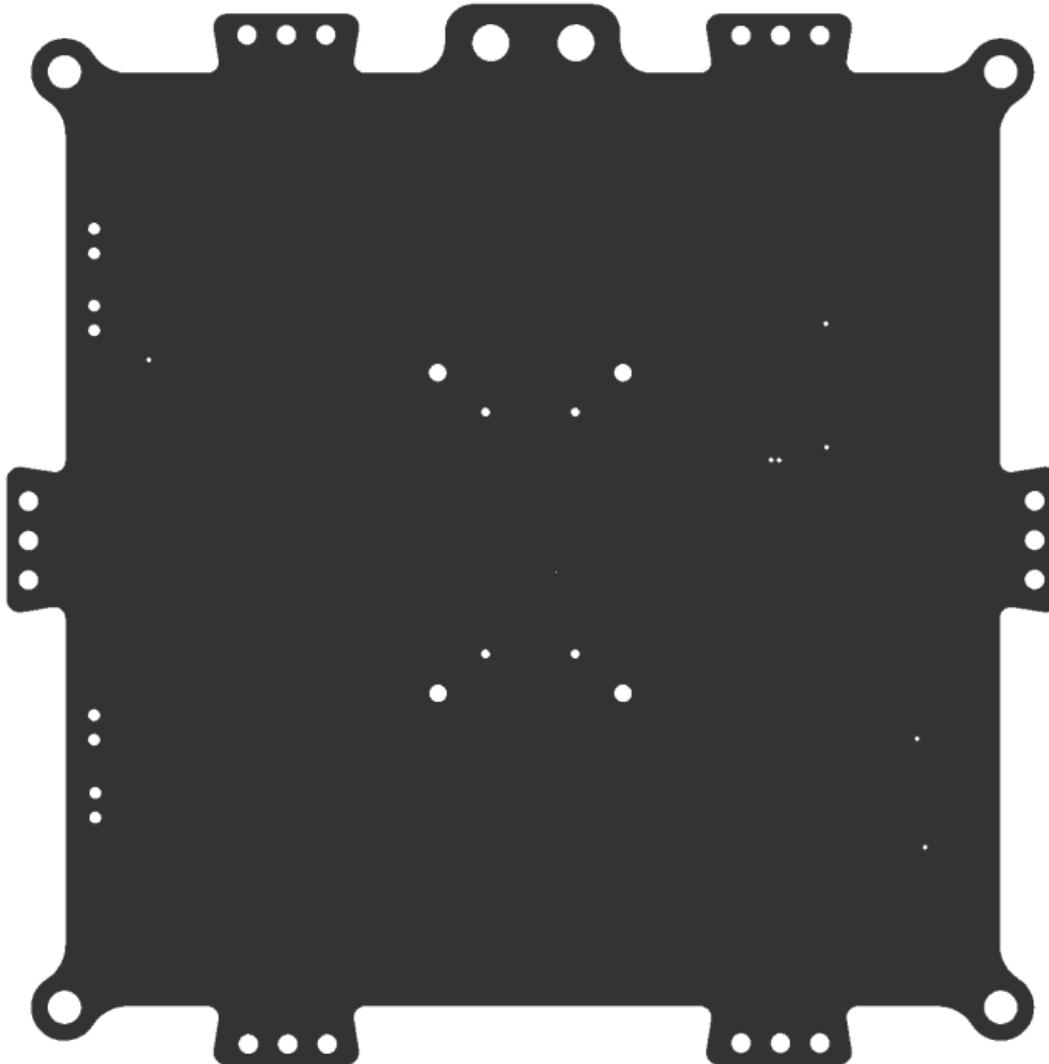
The Contoso Boards solution focuses on edge detection.

Camera

The following camera choices supported this workload:

- Camera placement: The camera is directly above at 90 degrees and about 16 inches from the part.
- Camera type: Since the conveyer system moves relatively slowly, the solution can use an area scan camera with a global shutter.
- Frame rate: For this use case, the camera captures about 30 frames per second.
- Resolution: The formula for required resolution is $\text{Res} = (\text{object size}) / (\text{details to capture})$. Based on this formula, $\text{Res} = 16''/8''$ gives 2 megapixels (MP) in x and 4MP in y , so Contoso Boards needs a camera capable of 4MP resolution.
- Sensor type: The targets aren't fast moving, and only require edge detection, so a CMOS sensor works well.
- Lighting: The solution use a white diffused filter back light. This lighting makes the part look almost black, with high contrast for edge detection.
- Color: Monochrome yields the sharpest edges for the AI detection model.

The following image shows what the camera captures in this scenario:



Hardware acceleration

Based on the workload, the use of TensorFlow, and the usage on several assembly lines, GPU-based hardware is the best choice for hardware acceleration.

ML model

The data scientists are most familiar with TensorFlow, so learning ONNX or other ML frameworks would slow down model development. [Azure Stack Edge](#) provides a centrally managed edge solution for all assembly lines.

User scenario 2: Safety

Contoso Shipping has had several pedestrian accidents at their loading docks. Most accidents happened when a truck left the loading dock, and the driver didn't see a dock worker walking in front of the truck. Contoso Shipping needed a vision AI solution that could watch for people, predict their direction of travel, and warn drivers of potential collisions.

Most of the data scientists at Contoso Shipping were familiar with [OpenVINO](#), and wanted to reuse the solution models on future hardware. The solution also needed to support power efficiency, and use the smallest possible number of cameras. Finally, Contoso Shipping wanted to manage the solution remotely for updates.

Cameras

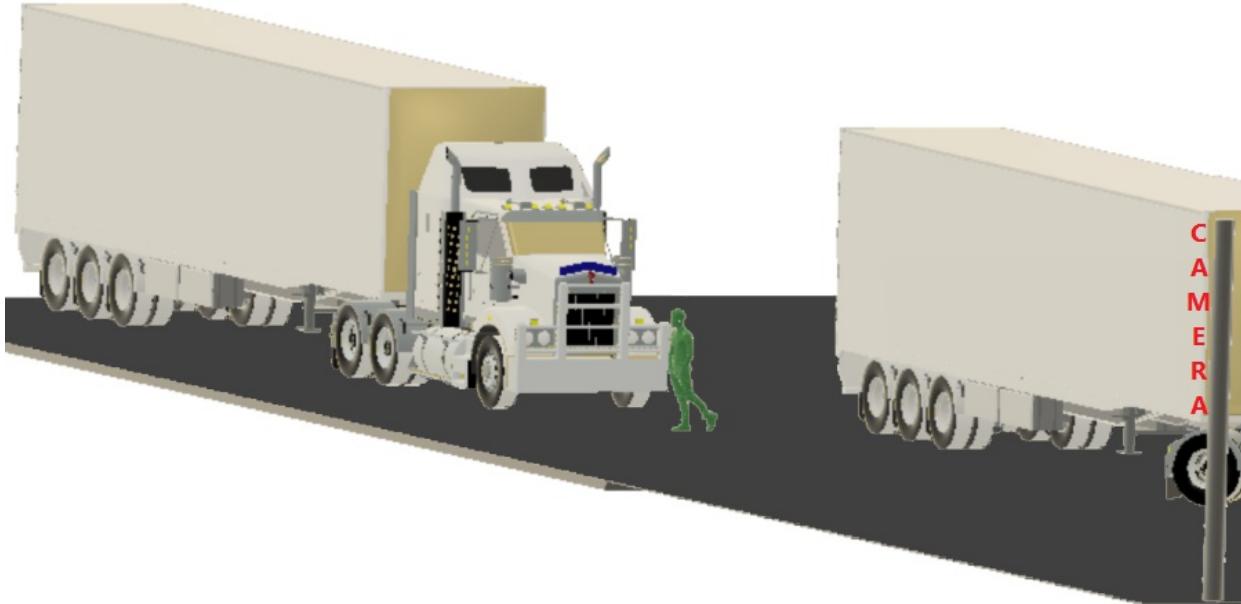
The solution uses 11 monochrome, 10MP CMOS cameras with IPX67 housings or weather boxes, mounted on 17-foot poles, 100 feet away from the trucks. The following sections describe how Contoso Shipping

determined these specifications.

Camera placement

Cameras needed to be 100 feet away from the fronts of the trucks. Camera focus had to be 10 feet in front of and behind the fronts of the trucks, giving a 20-foot depth of focus. Local zoning laws limited surveillance camera height to 20 feet.

The following illustration shows the camera placement for this scenario:



Resolution and field of view

The solution must capture only enough detail to detect a person in the frame. The pixels per foot (PPF) can be around 15-20, rather than the 80 PPF that facial recognition needs.

The formula for field of view (FOV) is $\text{FOV} = (\text{horizontal resolution}) / (\text{PPF})$. For resolution, the camera must use the right sensor for the use case.

This solution uses camera lenses that allow a 16-foot FOV. Using the preceding formula, a 16-foot FOV gives about 17.5 PPF, which falls within the required 15-20 PPF. This FOV means the solution should use 10MP cameras, which have a horizontal resolution of about 5184 pixels.

Since the cameras can look at a 16-foot path, a 165-foot long loading dock divided by a 16-foot FOV gives 10.3125 cameras. So the solution needs 11, 5184-horizontal pixel or 10MP cameras.

Sensor type

The cameras are outdoors, so the sensor type shouldn't allow *bloom*. Bloom is when light hits the sensor and overloads the sensor, causing overexposure or whiteout. CMOS is the sensor of choice.

Color and lighting

Contoso Shipping operates 24/7, and must also protect nighttime personnel. Monochrome handles low light conditions better than color. In this case, color information is unnecessary. Monochrome sensors are also lower cost.

ML model

Because the data scientists are familiar with OpenVINO, the solution builds data models in [ONNX](#).

Hardware acceleration

The distance from the cameras to the servers is too far for Gigabit Ethernet or USB connectivity, but there's a large mesh Wi-Fi network. The hardware must connect over Wi-Fi, and use as little power as possible.

Based on these requirements, the solution uses FPGA processors. The solution could also use ASIC processors,

but purpose-built ASIC chips don't meet the requirement for future usability.

Next steps

This series of articles described how to build a vision AI workload with Azure IoT Edge. For the other articles in this series, see:

- [Azure IoT Edge vision AI overview](#)
- [Camera selection for Azure IoT Edge vision AI](#)
- [Hardware acceleration in Azure IoT Edge vision AI](#)
- [Machine learning and data science in Azure IoT Edge vision AI](#)
- [Image storage and management for Azure IoT Edge vision AI](#)
- [Alert persistence in Azure IoT Edge vision AI](#)

To learn more about CNNs, vision AI, Azure Machine Learning, and Azure IoT Edge, see the following documentation:

- [Azure IoT Edge documentation](#)
- [Azure Machine Learning documentation](#)
- [Tutorial: Perform image classification at the edge with Custom Vision Service](#)
- [What is Computer Vision?](#)
- [What is Azure Video Analyzer? \(preview\)](#)
- [Azure Kinect DK developer kit documentation](#)
- [Open Neural Network Exchange \(ONNX\) ML framework](#)
- [Model management deep neural network \(MMDnn\) ML tool](#)

Related resources

For more computer vision architectures, examples, and ideas that use Azure IoT, see the following articles:

- [Getting started with Azure IoT solutions](#)
- [End-to-end manufacturing using computer vision on the edge](#)
- [Connected factory hierarchy service](#)
- [Connected factory signal pipeline](#)
- [Create smart places by using Azure Digital Twins](#)
- [Deploy AI and ML computing on-premises and to the edge](#)

Services in an Industrial IoT analytics solution

3/10/2022 • 19 minutes to read • [Edit Online](#)

Building on the architectural components in the recommended [Azure Industrial IoT analytics solution](#), this article discusses the subsystems and Azure services that can be used in such a solution. Your solution may not use all these services or may have additional services.

Time Series Service

A time series service will be required to provide a warm and cold store for the JSON-based industrial time series data. We recommend using [Time Series Insights \(or TSI\)](#) as the time series service for the following reasons:

- Ability to ingest streaming data directly from [Azure IoT Hub](#) or [Azure Event Hub](#).
- Multi-layered storage solution with warm and cold analytics support. It provides you the option to route data between warm and cold, for interactive analytics over warm data as well as operational intelligence over decades of historical data.
 - A highly interactive warm analytics solution to perform frequent, and large number of queries over shorter time span data.
 - A scalable, performant, and cost-optimized time series data lake based on Azure Storage allowing customers to trend years' worth of time series data.
- Asset hierarchy support that describes the domain and metadata associated with the derived and raw signals from industrial assets and devices. Multiple hierarchies can be defined reflecting different departments within your company. For instance, a factory hierarchy will look different from an Oil & Gas hierarchy.
- Flexible analytics platform to store historical time series data in your own Azure Storage account, thereby allowing you to have ownership of your industrial IoT data. Data is stored in open source Apache Parquet format that enables connectivity and interoperability across a variety of data scenarios including predictive analytics, machine learning, and other custom computations done using familiar technologies including Spark, Databricks, and Jupyter.
- Rich analytics with enhanced query APIs and user experience that combines asset-based data insights with rich, improvised data analytics with support for interpolation, scalar and aggregate functions, categorical variables, scatter plots, and time-shifting time series signals for in-depth analysis.
- Enterprise grade platform to support the scale, performance, security, and reliability needs of our industrial IoT customers.
- Ability to view and export the data in CSV format for further analysis.
- Ability to share analysis across your organization. Current visualizations can be saved and shared. Alternatively, the current state of each user's analysis is assigned a unique identifier. This identifier is placed in the URL allowing users to easily share their analysis by placing the URL in emails, documents, etc. Because TSI pricing is not *seat*-based, this democratizes the data by allowing anyone with access to the data to see it.

You can continue to use Time Series Insights with the old pricing model (S1, S2) for warm, unplanned data analysis. However, we highly encourage you to use the updated offering (PAYG) as it offers many new capabilities as well as Pay-As-You-Go pricing, cost savings, and flexibility to scale. Alternatively, [Azure Data Explorer](#) or [Cosmos DB](#) may be used as a time series database if you plan to develop a custom time series

service.

NOTE

When connecting Time Series Insights with IoT Hub or Event Hub, ensure you select an appropriate [Time Series ID](#). We recommend using a SCADA tag name field or OPC UA node id (for example, nsu=http://msft/boiler;i=#####) if possible, as these will map to leaf nodes in your [Time Series Model](#).

The data in Time Series Insights is stored in your [Azure Blob Storage account](#) (bring your own storage account) in Parquet file format. It is your data after all!

You can query your data in Time Series Insights using:

- [Time Series Insights Explorer](#)
- [Query API](#)
- [REST API](#)
- [Power BI](#)
- Any of your favorite BI and analytics tools (for example, Spark, Databricks, Azure Notebooks) by accessing the Parquet files in your Azure blob storage account

Microservices

Your IIoT analytics solution will require a number of microservices to perform functions such as:

- Providing HTTP REST APIs to support your web application.
 - We recommend creating HTTP-triggered [Azure Functions](#) to implement your APIs.
 - Alternatively, you can develop and host your REST APIs using Azure Service Fabric or [Azure Kubernetes Service \(AKS\)](#).
- Providing an HTTP REST API interface to your factory floor OPC UA servers (for example, using Azure Industrial IoT components consisting of OPC Publisher, OPC Twin and OPC Vault) to provide discovery, registration, and remote control of industrial devices.
 - For hosting the Azure Industrial IoT microservices, we recommend using Azure Kubernetes Service (AKS). See [Deploying Azure Industrial IoT Platform](#) to understand the various deployment options.
- Performing data transformation such as converting binary payloads to JSON or differing JSON payloads to a common, canonical format.
 - We recommend creating Azure Functions connected to IoT Hub to perform payload transformations.
 - Different industrial equipment vendors will send telemetry in different payload formats (JSON, binary, and so on) and schemas. When possible, we recommend converting the different equipment schemas to a common, canonical schema, ideally based on an industry standard.
 - If the message body is binary, use an Azure Function to convert the incoming messages to JSON and send the converted messages back to IoT Hub or to Event Hub.
 - When the message body is binary, [IoT Hub message routing](#) cannot be used against the message body, but can be used against [message properties](#).
 - The Azure Industrial IoT components include the capability to decode OPC UA binary messages to JSON.
- A [Data Ingest Administration](#) service for updating the list of tags monitored by your IIoT analytics solution.

- A [Historical Data Ingestion](#) service for importing historical data from your SCADA, MES, or historian into your solution.

Your solution will likely involve additional microservices to satisfy the specific requirements of your IIoT analytics solution. If your organization is new to building microservices, we recommend implementing custom microservices using Azure Functions. Azure Functions is an event-driven serverless compute platform that can be used to develop microservices and solve complex orchestration problems. It allows you to build and debug locally (in several software languages) without additional setup, deploy and operate at scale in the cloud, and integrate Azure services using triggers and bindings.

Both stateless and stateful microservices can be developed using Azure Functions. Azure Functions can use Cosmos DB, Table Storage, Azure SQL, and other databases to store stateful information.

Alternatively, if your organization has a previous experience building container-based microservices, we recommend you to also consider Azure Service Fabric or Azure Kubernetes Service (AKS). Refer to [Microservices in Azure](#) for more information.

Regardless of your microservices platform choice, we recommend using [Azure API Management](#) to create consistent and modern API gateways for your microservices. API Management helps abstract, publish, secure, and version your APIs.

Data Ingest Administration

We recommend developing a Data Ingest Administration service to add/update the list of tags monitored by your IIoT analytics solution.

SCADA *tags* are variables mapped to I/O addresses on a PLC or RTU. Tag names vary from organization to organization but often follow a naming pattern. As an example, tag names for a pump with tag number `14P103` located in STN001 (Station 001), has these statuses:

- STN001_14P103_RUN
- STN001_14P103_STOP
- STN001_14P103_TRIP

As new tags are created in your SCADA system, the IIoT analytics solution must become aware of these tags and subscribe to them in order to begin collecting data from them. In some cases, the IIoT analytics solution may not subscribe to certain tags as the data they contain may be irrelevant.

If your SCADA system supports OPC UA, new tags should appear as new NodeIDs in the OPC UA hierarchy. For example, the above tag names may appear as:

- ns=2;s= STN001_14P103_RUN
- ns=2;s= STN001_14P103_STOP
- ns=2;s= STN001_14P103_TRIP

We recommend developing a workflow that informs the administrators of the IIoT analytics solution when new tags are created, or existing tags are edited in the SCADA system. At the end of the workflow, the OPC Publisher is updated with the new/updated tags.

To accomplish this, we recommend developing a workflow that involves [Power Apps](#), [Logic Apps](#), and Azure Functions, as follows:

- The SCADA system operator can trigger the Logic Apps workflow using a Power Apps form whenever tags are created or edited in the SCADA system.
 - Alternatively, Logic Apps [connectors](#) can monitor a table in the SCADA system database for tag changes.
 - The OPC UA Discovery service can be used to both find OPC UA servers and the tags and methods

they implement.

- The Logic Apps workflow includes an approval step where the IIoT analytics solution owners can approve the new/updated tags.
- Once the new/updated tags are approved and a frequency assigned, the Logic App calls an Azure Function.
- The Azure function calls the OPC Twin microservice, which directs the OPC Publisher module to subscribe to the new tags.
 - A sample can be found [here](#).
 - If your solution involves third-party software, instead of OPC Publisher, configure the Azure Function to call an API running on the third-party software either directly or using an IoT Hub [Direct Method](#).

Alternatively, Microsoft Forms and Microsoft Flow can be used in place of Power Apps and Logic Apps.

Historical Data Ingestion

Years of historical data likely exists in your current SCADA, MES, or historian system. In most cases, you will want to import your historical data into your IIoT analytics solution.

Loading historical data into your IIoT analytics solution consists of three steps:

1. Export your historical data.
 - a. Most SCADA, MES, or historian systems have some mechanism that allows you to export your historical data, often as CSV files. Consult your system's documentation on how best to do this.
 - b. If there is no export option in your system, consult the system's documentation to determine if an API exists. Some systems support HTTP REST APIs or [OPC Historical Data Access \(HDA\)](#). If so, build an application or use a Microsoft partner solution that connects to the API, queries for the historical data, and saves it to a file in formats such as CSV, Parquet, TSV, and so on.
2. Upload to Azure.
 - a. If the aggregate size of the exported data is small, you can upload the files to Azure Blob Storage over the internet using [Azcopy](#).
 - b. If the aggregate size of the exported data is large (tens or hundreds of TBs), consider using [Azure Import/Export Service](#) or [Azure Data Box](#) to ship the files to the Azure region where your IIoT analytics solution is deployed. Once received, the files will be imported into your Azure Storage account.
3. Import your data.
 - a. This step involves reading the files in your Azure Storage account, serializing the data as JSON, and sending data as streaming events into Time Series Insights. We recommend using an Azure Function to perform this.
 - b. Time Series Insights only supports IoT Hub and Event Hub as data sources. We recommend using an Azure Function to send the events to a temporary Event Hub, which is connected to Time Series Insights.
 - c. Refer to [How to shape JSON events](#) and [Supported JSON shapes](#) for best practices on shaping your JSON payload.
 - d. Make sure to use the same [Time Series ID](#) as you do for your streaming data.
 - e. Once this process is completed, the Event Hub and Azure Function may be deleted. This is an optional step.

NOTE

Exporting large volumes of data from your industrial system (for example, SCADA or historian) may place a significant performance load on that system, which can negatively impact operations. Consider exporting smaller batches of historical data to minimize performance impacts.

Rules and Calculation Engine

Your IIoT analytics solution may need to perform near real-time (low latency) calculations and complex event processing (or CEP) over streaming data, before it lands in a database. For example, calculating moving averages or calculated *tags*. This is often referred to as a *calculations engine*. Your solution may also need to trigger actions (for example, display an alert) based on the streaming data. This is referred to as a *rules engine*.

We recommend using [Time Series Insights](#) for simple calculations, at query time. The [Time Series Model](#) introduced with Time Series Insights supports a number of formulas including: Avg, Min, Max, Sum, Count, First, and Last. The formulas can be created and applied using the [Time Series Insights APIs](#) or [Time Series Insights Explorer](#) user interface.

For example, a Production Manager may want to calculate the average number of widgets produced on a manufacturing line, over a time interval, to ensure productivity goals are met. In this example, we would recommend the Production Manager to use the Time Series Insights explorer interface to create and visualize the calculation. Or if you have developed a custom web application, it can use the Time Series Insights APIs to create the calculation, and the [Azure Time Series Insights JavaScript SDK \(or tsclient\)](#) to display the data in your custom web application.

For more advanced calculations and/or to implement a rules engine, we recommend using [Azure Stream Analytics](#). Azure Stream Analytics is a real-time analytics and complex event-processing engine, that is designed to analyze and process high volumes of fast streaming data from multiple sources simultaneously. Patterns and relationships can be identified in information extracted from a number of input sources including devices, sensors, click streams, social media feeds, and applications. These patterns can be used to trigger actions and initiate workflows such creating alerts, feeding information to a reporting tool, or storing transformed data for later use.

For example, a Process Engineer may want to implement a more complex calculation such as calculating the [standard deviation \(SDEV\)](#) of the widgets produced across a number of production lines to determine when any line is more than 2x beyond the mean over a period of time. In this example, we recommend using Stream Analytics, with a custom web application. The Process Engineer authors the calculations using the custom web application, which calls the [Stream Analytics REST APIs](#) to create and run these calculations (also known as *Jobs*). The Job output can be sent to an Event Hub, connected to Time Series Insights, so the result can be visualized in Time Series Insights explorer.

Similarly, for a *Rules Engine*, a custom web application can be developed that allows users to author alerts and actions. The web application creates associated Jobs in Azure Stream Analytics using the Steam Analytics REST API. To trigger actions, a Stream Analytics Job calls an Azure Function output. The Azure Function can call a Logic App or Power Automate task that sends an Email alert or invokes Azure SignalR to display a message in the web application.

Azure Stream Analytics supports processing events in CSV, JSON, and Avro data formats while Time Series Insights supports JSON. If your payload does not meet these requirements, consider using an Azure Function to perform data transformation prior to sending the data to Stream Analytics or Time Series Insights (using IoT Hub or Event Hubs).

Azure Stream Analytics also supports [reference data](#), a finite data set that is static or slowly changing in nature, used to perform a lookup or to augment your data streams. A common scenario is exporting asset metadata from your Enterprise Asset Management system and joining it with real-time data coming from those industrial devices.

Stream Analytics is also available as a [module](#) on the Azure IoT Edge runtime. This is useful for situations where complex event processing needs to happen at the Edge. As an alternative to Azure Stream Analytics, near real-time Calculation and Rules Engines may be implemented using [Apache Spark Streaming on Azure Databricks](#).

Notifications

Since the IIoT analytics solution is *not a control system*, it does not require a complete [Alarm Management](#) system. However, there will be cases where you will want the ability to detect conditions in the streaming data and generate notifications or trigger workflows. Examples include:

- temperature of a heat exchanger exceeding a configured limit, which changes the color of an icon in your web application,
- an error code sent from a pump, which triggers a work order in your ERP system, or
- the vibration of a motor exceeding limits, which triggers an email notification to an Operations Manager.

We recommend using Azure Stream Analytics to define and detect conditions in the streaming data (refer to the [rules engine](#) mentioned earlier). For example, a Plant Manager implements an automated workflow that runs whenever an error code is received from any equipment. In this example, your custom web application can use the [Stream Analytics REST API](#) to provide a user interface for the Plant Manager to create and run a job that monitors for specific error codes.

For defining an alert (email or SMS) or triggering a workflow, we recommend using Azure Logic Apps. Logic Apps can be used to build automated, scalable workflows, business processes, and enterprise orchestrations to integrate your equipment and data across cloud services and on-premises systems.

We recommend connecting Azure Stream Analytics with Azure Logic Apps using [Azure Service Bus](#). In the previous example, when an error code is detected by Stream Analytics, the job will send the error code to an Azure Service Bus queue output. A Logic App will be triggered to run whenever a message is received on the queue. This Logic App will then perform the workflow defined by the Plant Manager, which may involve creating a work order in Dynamics 365 or SAP, or sending an email to maintenance technician. Your web application can use the [Logic Apps REST API](#) to provide a user interface for the Plant Manager to author workflows or these can be built using the Azure portal authoring experience.

To display visual alerts in your web application, we recommend creating an Azure Stream Analytics job to detect specific events and send those to either:

- **An Event Hub output:** Then connect the Event Hub to Time Series Insights. Use the [Azure Time Series Insights JavaScript SDK \(tsiclient\)](#) to display the event in your web application.

or,

- **An Azure Functions output:** Then [develop an Azure Function](#) that sends the events to your web application using [SignalR](#).

Operational alarms and events triggered on premise can also be ingested into Azure for reporting and to trigger work orders, SMS messages, and emails.

Microsoft 365

The IIoT analytics solution can also include [Microsoft 365](#) services to automate tasks and send notifications. The following are a few examples:

- Receive email alerts in Microsoft Outlook or post a message to a Microsoft Teams channel when a condition is met in Azure Stream Analytics.
- Receive notifications as part of an approval workflow triggered by a Power App or Microsoft Forms submission.
- Create an item in a SharePoint list when an alert is triggered by a Logic App.
- Notify a user or execute a workflow when a new tag is created in a SCADA system.

Machine Learning

Machine learning models can be trained using your historical industrial data, enabling you to add predictive capabilities to your IIoT application. For example, your Data Scientists may be interested in using the IIoT analytics solution to build and train models that can predict events on the factory floor or indicate when maintenance should be conducted on an asset.

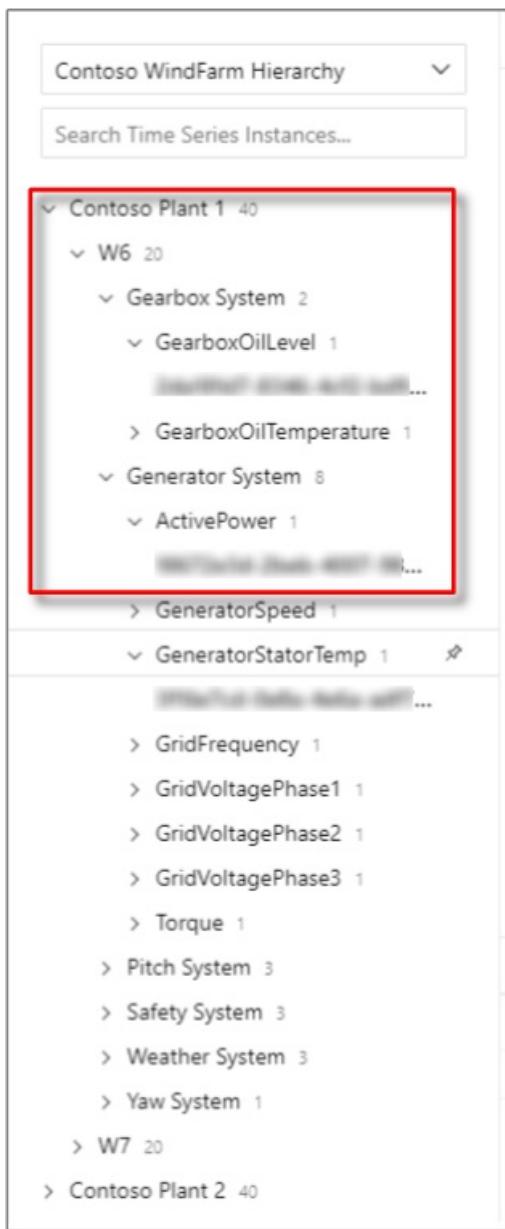
For building and training machine learning models, we recommend [Azure Machine Learning](#). Azure Machine Learning can [connect](#) to Time Series Insights data stored in your Azure Storage account. Using the data, you can create and train [forecasting models](#) in Azure Machine Learning. Once a model has been trained, it can be [deployed](#) as a web service on Azure (hosted on Azure Kubernetes Services or Azure Functions, for example) or to an Azure IoT Edge field gateway.

For those new to machine learning or organizations without Data Scientists, we recommend starting with [Azure Cognitive Services](#). Azure Cognitive Services are APIs, SDKs, and services available to help you build intelligent applications without having formal AI or data science skills or knowledge. Azure Cognitive Services enable you to easily add cognitive features into your IIoT analytics solution. The goal of Azure Cognitive Services is to help you create applications that can see, hear, speak, understand, and even begin to reason. The catalog of services within Azure Cognitive Services can be categorized into five main pillars - *Vision, Speech, Language, Web Search, and Decision*.

Asset Hierarchy

An asset hierarchy allows you to define hierarchies for classifying your asset, for example, Country > Location > Facility > Room. They may also contain the relationship between your assets. Many organizations maintain asset hierarchies within their industrial systems or within an Enterprise Asset Management (EAM) system.

The [Time Series Model](#) in Azure Time Series Insights provides asset hierarchy capabilities. Through the use of *Instances, Types* and *Hierarchies*, you can store metadata about your industrial devices, as shown in the image below.



If possible, we recommend exporting your existing asset hierarchy and importing it into Time Series Insights using the [Time Series Model APIs](#). We recommend periodically refreshing it as updates are made in your Enterprise Asset Management system.

In the future, asset models will evolve to become [digital twins](#), combining dynamic asset data (real-time telemetry), static data (3D models, metadata from Asset Management Systems), and graph-based relationships, allowing the digital twin to change in real-time along with the physical asset.

[Azure Digital Twins](#) is an Azure IoT service that provides the ability to:

- Create comprehensive models of physical environments,
- Create spatial intelligence graphs to model the relationships and interactions between people, places, and devices,
- Query data from a physical space rather than disparate sensors, and
- Build reusable, highly scalable, spatially aware experiences that link streaming data across the physical and digital world.

Business Process Integration

In some instances, you will want your IIoT analytics solution to perform actions based on insights from your industrial data. This can include raising alarms, sending email, sending SMS messages, or triggering a workflow in your line-of-business systems (for example, CRM, ERP, and so on). We recommend using Azure Logic Apps to

integrate your IIoT analytics solution with your line-of-business systems. Azure Logic Apps has a number of connectors to business systems and Microsoft services such as:

- Dynamics 365
- SharePoint Online
- Office 365 Outlook
- Salesforce
- SAP

For example, an error code from a pump is detected by an Azure Stream Analytics job. The job sends a message to Azure Service Bus and triggers a Logic App to run. The Logic App sends an email notification to the Plant Manager using the [Office 365 Outlook connector](#). It then sends a message to your SAP *S/4 HANA* system using the [SAP connector](#), which creates a Service Order in SAP.

User Management

User management involves managing user profiles and controlling what actions a user can perform in your IIoT analytics solution. For example, what asset data can a user view, or whether the user can create conditions and alerts. This is frequently referred to as role-based access control (RBAC).

We recommend implementing role-based access control using the [Microsoft identity platform](#) along with [Azure Active Directory](#). In addition, the Azure PaaS services mentioned in this IIoT analytics solution can integrate directly with Azure Active Directory, thereby ensuring security across your solution.

Your web application and custom microservices can also integrate with the Microsoft identity platform using libraries such as [Microsoft Authentication Library \(or MSAL\)](#) and protocols such as OAuth 2.0 and OpenID Connect.

User management also involves operations such as:

- creating a new user,
- updating a user's profile, such as their location and phone number,
- changing a user's password, and
- disabling a user's account.

For these operations, we recommend using the [Microsoft Graph](#).

Next steps

Data visualization is the backbone of a well-defined analytics system. Learn about the [data visualization techniques](#) that you can use with the IIoT analytics solution recommended in this series.

Architectural considerations in an IIoT analytics solution

3/10/2022 • 3 minutes to read • [Edit Online](#)

The [Microsoft Azure Well-Architected Framework](#) describes some key tenets of a good architectural design. Keeping in line with these tenets, this article describes the considerations in the reference [Azure Industrial IoT analytics solution](#) that improve its performance and resiliency.

Performance considerations

Azure PaaS services

All Azure PaaS services have an ability to scale up and/or out. Some services will do this automatically (for example, IoT Hub, Azure Functions in a Consumption Plan) while others can be scaled manually.

As you test your IIoT analytics solution, we recommend that you do the following:

- Understand how each service scales (and what the units of scale are).
- Collect performance metrics and establish baselines.
- Set up alerts when performance metrics exceed baselines.

All Azure PaaS services have a metrics blade that allows you to view service metrics, and configure conditions and alerts, which are collected and displayed in [Azure Monitor](#). We recommend enabling these features to ensure your solution performs as expected.

IoT Edge

Azure IoT Edge gateway performance is impacted by the following:

- The number of edge modules running and their performance requirements
- The number of messages processed by modules and EdgeHub
- Edge modules requiring GPU processing
- Offline buffering of messages
- The gateway hardware
- The gateway operating system

We recommend real world testing and/or testing with simulated telemetry to understand the field gateway hardware requirements for Azure IoT Edge. Conduct your initial testing using virtual machine where CPU, RAM, disk can be easily adjusted. Once approximate hardware requirements are known, get your field gateway hardware and conduct your testing again using actual hardware.

You should also test to ensure the following:

- No messages are being lost between source (for example, historian) and destination (for example, Time Series Insights).
- Acceptable message latency exists between the source and the destination.
- Source timestamps are preserved.
- Data accuracy is maintained, especially when performing data transformations.

Availability considerations

IoT Edge

A single Azure IoT Edge field gateway can be a single point of failure between your SCADA, MES, or historian and Azure IoT Hub. A failure can cause gaps in data in your IIoT analytics solution. To prevent this, IoT Edge can integrate with your on-premise Kubernetes environment, using it as a resilient, highly available infrastructure layer. For more information, see [How to install IoT Edge on Kubernetes \(Preview\)](#).

Network considerations

IoT Edge and firewalls

To maintain compliance with standards such as ISA 95 and ISA 99, industrial equipment is often installed in a closed Process Control Network (PCN), behind firewalls, with no direct access to the Internet (see [Purdue networking model](#)).

There are three options to connect to equipment installed in a PCN:

1. Connect to a higher-level system, such as a historian, located outside of the PCN.
2. Deploy an Azure IoT Edge device or virtual machine in a DMZ between the PCN and the internet.
 - a. The firewall between the DMZ and the PCN will need to allow inbound connections from the DMZ to the appropriate system or device in the PCN.
 - b. There may be no internal DNS setup to resolve PCN names to IP addresses.
3. Deploy an Azure IoT Edge device or virtual machine in the PCN and configure IoT Edge to communicate with the Internet through a Proxy server.
 - a. Additional IoT Edge setup and configuration are required. See [Configure an IoT Edge device to communicate through a proxy server](#).
 - b. The Proxy server may introduce a single point of failure and/or a performance bottleneck.
 - c. There may be no DNS setup in the PCN to resolve external names to IP addresses.

Azure IoT Edge will also require:

- access to container registries, such as Docker Hub or Azure Container Registry, to download modules over HTTPS,
- access to DNS to resolve external FQDNs, and
- ability to communicate with Azure IoT Hub using MQTT, MQTT over WebSockets, AMQP, or AMQP over WebSockets.

For additional security, industrial firewalls can be configured to only allow traffic between IoT Edge and IoT Hub using [Service Tags](#). IP address prefixes of IoT Hub public endpoints are published periodically under the [AzureIoTHub](#) service tag. Firewall administrators can programmatically retrieve the current list of service tags, together with IP address range detail, and update their firewall configuration.

Next steps

- For a more detailed discussion of the recommended architecture and implementation choices, download and read the [Microsoft Azure IoT Reference Architecture pdf](#).
- [Azure Industrial IoT components, tutorials, and source code](#).
- For detailed documentation of the various Azure IoT services, see [Azure IoT Fundamentals](#).

Azure mainframe and midrange architecture concepts and patterns

3/10/2022 • 12 minutes to read • [Edit Online](#)

Mainframe and midrange hardware is composed of a family of systems from various vendors (all with a history and goal of high performance, high throughput, and sometimes high availability). These systems were often *scale-up* and monolithic, meaning they were a single, large frame with multiple processing units, shared memory, and shared storage.

On the application side, programs were often written in one of two flavors: either transactional or batch. In both cases, there were a variety of programming languages that were used, including COBOL, PL/I, Natural, Fortran, REXX, and so on. Despite the age and complexity of these systems, there are many migration pathways to Azure.

On the data side, data is usually stored in files and in databases. Mainframe and midrange databases commonly come in a variety of possible structures, such as relational, hierarchical, and network, among others. There are different types of file organizational systems, where some of them can be indexed and can act as a key-value stores. Further, data encoding in mainframes can be different from the encoding usually handled in non-mainframe systems. Therefore, data migrations should be handled with upfront planning. There are many options for migrating to the Azure data platform.



Migrating legacy systems to Azure

In many cases, mainframe, midrange, and other server-based workloads can be replicated in Azure with little to no loss of functionality. Sometimes users do not notice changes in their underlying systems. In other situations, there are options for refactoring and re-engineering the legacy solution into an architecture that is in alignment with the cloud. This is done while still maintaining the same or very similar functionality. The architectures in this content set (plus the additional white papers and other resources provided below) will help guide you through this process.

Mainframe and midrange concepts

In our mainframe architectures, we use the following terms.

Mainframes

Mainframes were designed as scale-up servers to run high-volume online transactions and batch processing in the late 1950s. As such, mainframes have software for online transaction forms (sometimes called green screens) and high-performance I/O systems, for processing the batch runs. Mainframes have a reputation for high reliability and availability, in addition to their ability to run online and batch jobs.

Mainframe storage

Part of demystifying mainframes involves decoding various overlapping terms. For example, central storage, real memory, real storage, and main storage generally all refer to storage that is attached directly to the

mainframe processor. Mainframe hardware includes processors and many other devices, such as direct-access storage devices (DASDs), magnetic tape drives, and several types of user consoles. Tapes and DASDs are used for system functions and by user programs.

Types of physical storage:

- **Central storage.** Located directly on the mainframe processor, it's also known as *processor storage* or *real storage*.
- **Auxiliary storage.** Located separately from the mainframe, it includes storage on DASDs, which is also known as *paging storage*.

MIPS

The measurement of millions of instructions per second (MIPS) provides a constant value of the number of cycles per second, for a given machine. MIPS are used to measure the overall compute power of a mainframe. Mainframe vendors charge customers, based on MIPS usage. Customers can increase mainframe capacity to meet specific requirements. IBM maintains a [processor capacity index](#), which shows the relative capacity across different mainframes.

The table below shows typical MIPS thresholds across small, medium, and large enterprise organizations (SORGs, MORGs, and LORGs).

CUSTOMER SIZE	TYPICAL MIPS USAGE
SORG	Less than 500 MIPS
MORG	500 MIPS to 5,000 MIPS
LORG	More than 5,000 MIPS

Mainframe data

Mainframe data is stored and organized in a variety of ways, from relational and hierarchical databases to high throughput file systems. Some of the common data systems are z/OS Db2 for relational data and IMS DB for hierarchical data. For high throughput file storage, you might see VSAM (IBM Virtual Storage Access Method). The following table provides a mapping of some of the more common mainframe data systems, and their possible migration targets into Azure.

DATA SOURCE	TARGET PLATFORM IN AZURE
z/OS Db2 & Db2 LUW	Azure SQL DB, SQL Server on Azure VMs, Db2 LUW on Azure VMs, Oracle on Azure VMs, Azure Database for PostgreSQL
IMS DB	Azure SQL DB, SQL Server on Azure VMs, Db2 LUW on Azure VMs, Oracle on Azure VMs, Azure Cosmos DB
Virtual Storage Access Method (VSAM), Indexed Sequential Access Method (ISAM), other flat files	Azure SQL DB, SQL Server on Azure VMs, Db2 LUW on Azure VMs, Oracle on Azure VMs, Azure Cosmos DB
Generation Date Groups (GDGs)	Files on Azure using extensions in the naming conventions to provide similar functionality to GDGs

Midrange systems, Unix variants, and other legacy systems

Midrange systems and midrange computers are loosely defined terms for a computer system that is more powerful than a general-purpose personal computer, but less powerful than a full-size mainframe computer. In most instances, a midrange computer is employed as a network server, when there are a small to medium

number of client systems. The computers generally have multiple processors, a large amount of random access memory (RAM), and large hard drives. Additionally, they usually contain hardware that allows for advanced networking, and ports for connecting to more business-oriented peripherals (such as large-scale data storage devices).

Common systems in this category include AS/400 and the IBM i and p series. Unisys also has a collection of midrange systems.

Unix operating system

The Unix operating system was one of the first enterprise-grade operating systems. Unix is the base operating system for Ubuntu, Solaris, and operating systems that follow POSIX standards. Unix was developed in the 1970s by Ken Thompson, Dennis Ritchie, and others at AT&T Laboratories. It was originally meant for programmers who are developing software, rather than non-programmers. It was distributed to government organizations and academic institutions, both of which led Unix to being ported to a wider variety of variations and forks, with different specialized functions. Unix and its variants (such as AIX, HP-UX, and Tru64) are commonly found running on legacy systems, such as IBM mainframes, AS/400 systems, Sun Sparc, and DEC hardware-based systems.

Other systems

Other legacy systems include the family of systems from Digital Equipment Corporation (DEC), such as the DEC VAX, DEC Alpha, and DEC PDP. The DEC systems initially ran the VAX VMS operating system, then eventually they moved to Unix variants, such as Tru64. Other systems include ones that are based on the PA-RISC architecture, such as the HP-3000 and HP-9000 systems.

Midrange data and storage

Midrange data is stored and organized in variety of ways, from relational and hierarchical databases, to high throughput file systems. Some of the common data systems are Db2 for i (for relational data), and IMS DB for hierarchical data. The following table provides a mapping of some of the more common mainframe data systems and the possible migration targets into Azure.

DATA SOURCE	TARGET PLATFORM IN AZURE
Db2 for i	Azure SQL DB, SQL Server on Azure VMs, Azure Database for PostgreSQL, Db2 LUW on Azure VMs, Oracle on Azure VMs
IMS DB	Azure SQL DB, SQL Server on Azure VMs, Db2 LUW on Azure VMs, Oracle on Azure VMs, Azure Cosmos DB

Endianness

Consider the following details about endianness:

- RISC and x86 processors differ in *endianness*, a term used to describe how a system stores bytes in computer memory.
- RISC-based computers are known as big endian systems, because they store the most significant ("big") value first—that is, in the lowest storage address.
- Most Linux computers are based on the x86 processor, which are little endian systems, meaning they store the least significant ("little") value first.

The following figure visually shows you the difference between big endian and little endian.



Big-Endian

0xBEBAFECA
will be stored as
BE | BA | FE | CA



Little-Endian

0xBEBAFECA
will be stored as
CA | FE | BA | BE

High-level architectural types

Rehost

Often referred to as a lift-and-shift migration, this option doesn't require code changes. You can use it to quickly migrate your existing applications to Azure. Each application is migrated as is, to reap the benefits of the cloud (without the risk and cost that are associated with code changes).

Rehost architectures

-

[Stromasys Charon-SSP Solaris emulator on Azure VMs](#)

- 09/09/2020
- 4 min read

Charon-SSP cross-platform hypervisor emulates legacy Sun SPARC systems on industry standard x86-64 computer systems and VMs.

-

[Migrate IBM mainframe apps to Azure with TmaxSoft OpenFrame](#)

- 09/18/2020
- 5 min read

Migrate IBM zSeries mainframe applications to Azure. Use a no-code approach that TmaxSoft OpenFrame offers for this lift and shift operation.

-

[Unisys ClearPath Forward mainframe rehost to Azure using Unisys virtualization](#)

- 06/28/2021
- 6 min read

-

[Using LzLabs Software Defined Mainframe \(SDM\) in an Azure VM deployment](#)

- 06/28/2021

- 7 min read

An approach for rehosting mainframe legacy applications in Azure using the LzLabs Software Defined Mainframe platform.

Refactor

Refactoring requires minimal changes to applications. This often enables the application architecture to leverage Azure platform as a service (PaaS) and to leverage additional cloud offerings. For example, you could migrate compute components of existing applications to Azure App Service or to Azure Kubernetes Service (AKS). You could also refactor relational and nonrelational databases into various options, such as Azure SQL Managed Instance, Azure Database for MySQL, Azure Database for PostgreSQL, and Azure Cosmos DB.

Refactor architectures

- [\[Refactor architectures\]](#)

[General mainframe refactor to Azure](#)

- 08/10/2021
- 8 min read

See how to refactor general mainframe applications to run more cost-effectively and efficiently on Azure.

- [\[Refactor architectures\]](#)

[Micro Focus Enterprise Server on Azure VMs](#)

- 03/01/2021
- 6 min read

Optimize, modernize, and streamline IBM z/OS mainframe applications by using Micro Focus Enterprise Server 6.0 on Azure VMs.

- [\[Refactor architectures\]](#)

[Refactor IBM z/OS mainframe Coupling Facility \(CF\) to Azure](#)

- 12/01/2020
- 7 min read

Learn how Azure services and components can provide scale-out performance comparable to IBM z/OS mainframe CF and Parallel Sysplex capabilities.

- [\[Refactor architectures\]](#)

[Unisys Dorado mainframe migration to Azure with Astadia & Micro Focus](#)

- 03/19/2021
- 9 min read

Migrate Unisys Dorado mainframe systems with Astadia and Micro Focus products. Move to Azure without rewriting code, switching data models, or updating screens.

- [\[Refactor architectures\]](#)

[Unisys mainframe migration](#)

- 11/17/2020
- 7 min read

Learn about options for using the Asysco Automated Migration Technology (AMT) Framework to migrate Unisys mainframe workloads to Azure.

- [REDACTED]

[IBM System i \(AS/400\) to Azure using Infinite i](#)

- 05/14/2021
- 7 min read

Use Infinite i to easily migrate your IBM System i (AS/400) workloads to Azure. You can lower costs, improve performance, improve availability, and modernize.

- [REDACTED]

[IBM z/OS mainframe migration with Asysco](#)

- 04/01/2021
- 7 min read

See how to use the Asysco Automated Migration Technology (AMT) framework to migrate IBM z/OS mainframe workloads to Azure.

- [REDACTED]

[Refactor mainframe applications to Azure with Raincode compilers](#)

- 06/25/2021
- 7 min read

This architecture shows how the Raincode COBOL compiler modernizes mainframe legacy applications.

- [REDACTED]

[IBM z/OS online transaction processing on Azure](#)

- 07/30/2021
- 8 min read

Migrate a z/OS online transaction processing (OLTP) workload to an Azure application that is cost-effective, responsive, scalable, and adaptable.

Re-engineer

Re-engineering for migration focuses on modifying and extending application functionality and the code base to optimize the application architecture for cloud scalability. For example, you could break down a monolithic application into a group of microservices that work together and scale easily. You could also rearchitect relational and nonrelational databases to a fully managed database solution, such as SQL Managed Instance, Azure Database for MySQL, Azure Database for PostgreSQL, and Azure Cosmos DB.

Re-engineer architectures

- [REDACTED]

[High-volume batch transaction processing](#)

- 03/22/2021
- 6 min read

Use Azure Kubernetes Service (AKS) and Azure Service Bus to implement high-volume batch transaction processing.

-

[Integrate IBM mainframe and midrange message queues with Azure](#)

- 06/02/2021
- 7 min read

This example describes a data-first approach to middleware integration that enables IBM message queues (MQs).

-

[Re-engineer IBM z/OS batch applications on Azure](#)

- 8/13/2021
- 9 min read

Use Azure services to re-engineer mainframe batch applications. This architecture change can reduce costs and improve scalability.

Dedicated hardware

Another pattern for migrations into Azure (for legacy systems) is what is known as *dedicated hardware*. This pattern is where legacy hardware (such as IBM Power Systems) runs inside the Azure datacenter, with an Azure managed-service wrapping around the hardware, which enables easy cloud management and automation. Further, this hardware is available to connect to and use with other Azure IaaS and PaaS services.

Dedicated hardware architectures

-

[Migrate AIX workloads to Skytap on Azure](#)

- 05/24/2021
- 5 min read

This example illustrates a migration of AIX logical partitions (LPARs) to Skytap on Azure.

-

[Migrate IBM i series applications to Skytap on Azure](#)

- 05/24/2021
- 5 min read

This example architecture shows how to use the native IBM i backup and recovery services with Microsoft Azure components.

Data movement and migration

A key part of legacy migrations and transformations to Azure is consideration for data. This can include not only

data movement, but also data replication and synchronization.

Data movement and migration architectures

- [Redacted]

[Modernize mainframe & midrange data](#)

- 11/04/2020
- 7 min read

Learn how to modernize IBM mainframe and midrange data. See how to use a data-first approach to migrate this data to Azure.

- [Redacted]

[Replicate and sync mainframe data in Azure](#)

- 12/29/2020
- 8 min read

Replicate data while modernizing mainframe and midrange systems. Sync on-premises data with Azure data during modernization.

- [Redacted]

[Mainframe access to Azure databases](#)

- 05/03/2021
- 4 min read

Give mainframe applications access to Azure data without changing code. Use Microsoft Service for DRDA to run Db2 SQL statements on a SQL Server database.

- [Redacted]

[Mainframe file replication and sync on Azure](#)

- 04/01/2021
- 5 min read

Learn about several options for moving, converting, transforming, and storing mainframe and midrange file system data on-premises and in Azure.

Next steps

- For more information, please contact legacy2azure@microsoft.com.
- See the [Microsoft Azure Well-Architected Framework](#).

Related resources

The white papers, blogs, webinars, and other resources are available to help you on your journey, to understand the pathways to migrate legacy systems into Azure:

Whitepapers

- [Stromasys Charon-SSP Solaris Emulator: Azure Setup Guide](#)

- Stromasys legacy server emulation on Azure: Running applications designed for SPARC, Alpha, VAX, PDP-11, and HP 3000
- Deploy Db2 pureScale on Azure (Whitepaper)
- Install IBM DB2 pureScale on Azure (Azure Docs)
- Demystifying mainframe to Azure migration
- Microsoft Azure Government cloud for mainframe applications
- Set up Micro Focus Enterprise Server 4.0 and Enterprise Developer 4.0 in Azure
- Set up IBM Z Development and Test Environment 12.0 in Azure
- Move mainframe compute and storage to Azure
- E-Book: Install TmaxSoft OpenFrame on Azure

Webinars

- Angelbeat - Retail Industry Legacy Webinar
- Mainframe Transformation to Azure
- Mainframe Transformation: Azure is the New Mainframe
- ClearPath MCP Software Series For Azure
- Leverage the Power of Azure with Steve Read
- Carahsoft - Monolithic Mainframe to Azure Gov Cloud The USAF Journey
- Carahsoft - Topics in Government Mainframe Transformation to Azure Gov Cloud
- Skytap on Azure Webinar
- Bridge to Application Modernization: Virtualized SPARC/PA-RISK/DEC to Azure

Blog posts

- Running Micro Focus Enterprise Server 4.0 in a Docker Container in Azure
- Deploy Micro Focus Enterprise Server 4.0 to AKS
- Migrating iSeries (AS/400) Legacy Applications to Azure
- Migrating iSeries (AS/400) Legacy Applications to Azure with Infinite
- Migrating AIX Workloads to Azure: Approaches and Best Practices
- Using Containers for Mainframe Modernization
- Deploying NTT Data UniKix in Azure, Part 1 Deploying the VM
- MIPS Equivalent Sizing for IBM CICS COBOL Applications Migrated to Microsoft Azure
- Set up Micro Focus Enterprise Server 4.0 and Enterprise Developer 4.0 in Azure
- Set up IBM Z Development and Test Environment 12.0 in Azure

Customer stories

Different industries are migrating from legacy mainframe and midrange systems in innovative and inspiring ways. Following are a number of customer case studies and success stories:

- Mainframe to Azure: A Real World Modernization Case Study (GEICO and AIS)
- Jefferson County, Alabama
- Customer Technical Story: Actuarial Services Company - DEC Alpha to Azure using Stromasys
- Astadia & USAF Complete Mission-Critical Mainframe-to-Cloud Migration | Business Wire
- United States Air Force | Case Study (astadia.com)

Networking architecture design

3/10/2022 • 3 minutes to read • [Edit Online](#)

This article provides information about sample architectures, solutions, and guides that can help you explore networking in Azure.

Designing and implementing Azure networking capabilities is a critical part of your cloud solution. You'll need to make networking design decisions to properly support your workloads and services.

Azure provides a wide range of networking tools and capabilities. These are just some of the key networking services available in Azure:

- [Azure Virtual Network](#). Provision private networks, and optionally connect to on-premises datacenters.
- [Azure Virtual WAN](#). Optimize and automate branch-to-branch connectivity.
- [Azure Private Link](#). Enable private access to services that are hosted on the Azure platform while keeping your data on the Microsoft network.
- [Azure Firewall](#). Provide protection for your Azure Virtual Network resources.
- [Azure Application Gateway](#). Build highly secure, scalable, highly available web front ends.
- [Azure ExpressRoute](#). Create a fast, reliable, and private connection to Azure.
- [Azure Load Balancer](#). Deliver high availability and network performance to your apps.
- [Azure VPN Gateway](#). Establish high security cross-premises connectivity.

For information about more Azure networking services, see [Azure networking](#).

Introduction to networking on Azure

If you're new to networking on Azure, the best way to learn more is with [Microsoft Learn](#), a free online training platform. Microsoft Learn provides interactive training for Microsoft products and more.

Here's a good introduction to Azure networking:

- [Explore Azure networking services](#)

And here's a comprehensive learning path:

- [Configure and manage virtual networks for Azure administrators](#)

Path to production

Consider these technologies and solutions as you plan and implement your deployment:

- [Azure Firewall architecture overview](#)
- [Azure Private Link in a hub-and-spoke network](#)
- [Build solutions for high availability by using availability zones](#)
- [Add IP address spaces to peered virtual networks](#)
- [Choose between virtual network peering and VPN gateways](#)
- [Use Azure ExpressRoute with Microsoft Power Platform](#)

Best practices

The [Azure Well-Architected Framework](#) is a set of guiding tenets, based on five pillars, that you can use to

improve the quality of your architectures. These articles apply the pillars to the use of some Azure networking services:

- [Review of Azure Application Gateway](#)
- [Review of Azure Firewall](#)
- [Review of an Azure NAT gateway](#)

The [Cloud Adoption Framework](#) is a collection of documentation, implementation guidance, best practices, and tools that are designed to accelerate your cloud adoption. You might find these articles helpful as you plan and implement your networking solution:

- [Connectivity to other cloud providers](#)
- [Connectivity to Oracle Cloud Infrastructure](#)

Networking architectures

The following sections, organized by category, provide links to sample networking architectures.

High availability

- [Deploy highly available NVAs](#)
- [High availability and disaster recovery scenarios for IaaS apps](#)
- [Multi-tier web application built for HA/DR](#)

Hybrid networking

- [Azure Automation update management](#)
- [Connect standalone servers by using Azure Network Adapter](#)
- [Design a hybrid Domain Name System solution with Azure](#)
- [Hybrid availability and performance monitoring](#)
- [Implement a secure hybrid network](#)

Hub-and-spoke topology

- [Hub-and-spoke network topology in Azure](#)
- [Hub-and-spoke network topology with Azure Virtual WAN](#)

Virtual WAN

- [Global transit network architecture and Virtual WAN](#)
- [Interconnect with China using Azure Virtual WAN and Secure Hub](#)
- [Migrate to Azure Virtual WAN](#)
- [SD-WAN connectivity architecture with Azure Virtual WAN](#)
- [Virtual WAN network topology \(Microsoft-managed\)](#)
- [Virtual WAN architecture optimized for department-specific requirements](#)
- [Hub-and-spoke network topology with Azure Virtual WAN](#)

Multi-region networking

- [Multi-region N-tier application](#)
- [Multi-region load balancing with Azure Traffic Manager and Application Gateway](#)

Stay current with networking

Get the [latest updates on Azure networking products and features](#).

Additional resources

Example solutions

These are some additional sample networking architectures:

- [Traditional Azure networking topology](#)
- [What is an Azure landing zone?](#)
- [Multitenant SaaS on Azure](#)
- [Network-hardened web application with private connectivity to PaaS datastores](#)
- [Network topology and connectivity for Azure VMware Solution](#)
- [Private Link and DNS integration at scale](#)
- [Trusted Internet Connection \(TIC\) 3.0 compliance for internet-facing applications](#)
- [Update route tables by using Azure Route Server](#)

AWS or Google Cloud professionals

These articles provide service mapping and comparison between Azure and other cloud services. They can help you ramp up quickly on Azure.

- [Compare AWS and Azure networking options](#)
- [Google Cloud to Azure services comparison - Networking](#)

Azure Well-Architected Framework review of Azure Application Gateway

3/10/2022 • 14 minutes to read • [Edit Online](#)

This article provides architectural best practices for the Azure Application Gateway v2 family of SKUs. The guidance is based on the five pillars of architecture excellence: Cost Optimization, Operational Excellence, Performance Efficiency, Reliability, and Security.

We assume that you have working knowledge of Azure Application Gateway and are well versed with v2 SKU features. As a refresher, review the full set of [Azure Application Gateway features](#).

Cost Optimization

Review and apply the [cost principles](#) when making design choices. Here are some best practices.

Review Application Gateway pricing

Familiarize yourself with Application Gateway pricing to help you identify the right deployment configuration for your environment. Ensure that the options are adequately sized to meet the capacity demand and deliver expected performance without wasting resources.

For information about Application Gateway pricing, see [Understanding Pricing for Azure Application Gateway and Web Application Firewall](#).

Use these resources to estimate cost based on units of consumption.

- [Azure Application Gateway pricing](#)
- [Pricing calculator](#)

Review underutilized resources

Identify and delete Application Gateway instances with empty backend pools.

Stop Application Gateway instances when not in use

You aren't billed when Application Gateway is in the stopped state.

Continuously running Application Gateway instances can incur extraneous costs. Evaluate usage patterns and stop instances when you don't need them. For example, usage after business hours in Dev/Test environments is expected to be low.

See these articles for information about how to stop and start instances.

- [Stop-AzApplicationGateway](#)
- [Start-AzApplicationGateway](#)

Have a scale-in and scale-out policy

A scale-out policy ensures that there will be enough instances to handle incoming traffic and spikes. Also, have a scale-in policy that makes sure the number of instances are reduced when demand drops. Consider the choice of instance size. The size can significantly impact the cost. Some considerations are described in the [Estimate the Application Gateway instance count](#).

For more information, see [Autoscaling and Zone-redundant Application Gateway v2](#).

Review consumption metrics across different parameters

You're billed based on metered instances of Application Gateway based on the metrics tracked by Azure. Here's an example of cost incurred view in [Azure Cost Management + Billing](#).

The example is based on the current price and is subject to change. This is shown for information purposes only.

The screenshot shows the Azure Cost Management portal interface. At the top, it displays the scope as 'rg-bu0001a0008', the view as 'Cost by resource', the date as 'Jun 2021', and an 'Add filter' button. Below this, the actual cost is listed as '\$82.55'. The forecast chart and budget status are shown as '--'. A 'Filter items' dropdown is open, showing 10 rows of data. The first row is expanded to show detailed metrics for an application gateway named 'apw-aks-u4lhb2tam5j4s'. The metrics table includes columns for Service name, Meter, and Cost (with an up/down arrow). The data shows three capacity units costing \$40.60 and one bandwidth unit costing \$0.83. Other resources listed include Log Analytics workspace, Container registry, Private endpoint, and another private endpoint.

Evaluate the various metrics and capacity units and determine the cost drivers.

These are key metrics for Application Gateway. This information can be used to validate that the provisioned instance count matches the amount of incoming traffic.

- Estimated Billed Capacity Units
- Fixed Billable Capacity Units
- Current Capacity Units

For more information, see [Application Gateway metrics](#).

Make sure you account for bandwidth costs. For details, see [Traffic across billing zones and regions](#).

Performance Efficiency

Take advantage features for autoscaling and performance benefits

The v2 SKU offers autoscaling to ensure that your Application Gateway can scale up as traffic increases. When compared to v1 SKU, v2 has capabilities that enhance the performance of the workload. For example, better TLS offload performance, quicker deployment and update times, zone redundancy, and more. For more information about autoscaling features, see [Autoscaling and Zone-redundant Application Gateway v2](#).

If you are running v1 SKU gateways, consider migrating to v2 SKU. See [Migrate Azure Application Gateway and Web Application Firewall from v1 to v2](#).

General best practices related to Performance Efficiency are described in [Performance efficiency principles](#).

Estimate the Application Gateway instance count

Application Gateway v2 scales out based on many aspects, such as CPU, memory, network utilization, and more. To determine the approximate instance count, factor in these metrics:

- **Current compute units**—Indicates CPU utilization. 1 Application Gateway instance is approximately 10 compute units.

- **Throughput**—Application Gateway instance can serve 60-75 Mbps of throughput. This data depends on the type of payload.

Consider this equation when calculating instance counts.

$$\text{Approximate instance count} = \max \left\{ \frac{\text{Current compute units}}{10}, \frac{\text{Throughput in Mbps}}{60} \right\}$$

After you've estimated the instance count, compare that value to the maximum instance count. This will indicate how close you are to the maximum available capacity.

Define the minimum instance count

For Application Gateway v2 SKU, autoscaling takes some time (approximately six to seven minutes) before the additional set of instances is ready to serve traffic. During that time, if there are short spikes in traffic, expect transient latency or loss of traffic.

We recommend that you set your minimum instance count to an optimal level. After you estimate the average instance count and determine your Application Gateway autoscaling trends, define the minimum instance count based on your application patterns. For information, see [Application Gateway high traffic support](#).

Check the **Current Compute Units** for the past one month. This metric represents the gateway's CPU utilization. To define the minimum instance count, divide the peak usage by 10. For example, if your average **Current Compute Units** in the past month is 50, set the minimum instance count to 5.

Define the maximum instance count

We recommend 125 as the maximum autoscale instance count. Make sure the subnet that has the Application Gateway has sufficient available IP addresses to support the scale-up set of instances.

Setting the maximum instance count to 125 has no cost implications because you're billed only for the consumed capacity.

Define Application Gateway subnet size

Application Gateway needs a dedicated subnet within a virtual network. The subnet can have multiple instances of the deployed Application Gateway resource. You can also deploy other Application Gateway resources in that subnet, v1 or v2 SKU.

Here are some considerations for defining the subnet size:

- Application Gateway uses one private IP address per instance and another private IP address if a private front-end IP is configured.
- Azure reserves five IP addresses in each subnet for internal use.
- Application Gateway (Standard or WAF SKU) can support up to 32 instances. Taking 32 instance IP addresses + 1 private front-end IP + 5 Azure reserved, a minimum subnet size of /26 is recommended. Because the Standard_v2 or WAF_v2 SKU can support up to 125 instances, using the same calculation, a subnet size of /24 is recommended.
- If you want to deploy additional Application Gateway resources in the same subnet, consider the additional IP addresses that will be required for their maximum instance count for both, Standard and Standard v2.

Operational Excellence

Monitoring and diagnostics are crucial. Not only can you measure performance statistics but also use metrics to troubleshoot and remediate issues quickly.

Monitor capacity metrics

Use these metrics as indicators of utilization of the provisioned Application Gateway capacity. We strongly

recommend setting up alerts on capacity. For details, see [Application Gateway high traffic support](#).

METRIC	DESCRIPTION	USE CASE
Current Compute Units	CPU utilization of virtual machine running Application Gateway. One Application Gateway instance supports 10 Compute Units.	Helps detect issues when more traffic is sent than what Application Gateway instances can handle.
Throughput	Amount of traffic (in Bps) served by Application Gateway.	This threshold is dependent on the payload size. For smaller payloads but more frequent connections, expect lower throughput limits and adjust alerts accordingly.
Current Connections	Active TCP connections on Application Gateway.	Helps detect issues where the connection count increases beyond the capacity of Application gateway. Look for a drop in capacity unit when the connection count increases, look for a simultaneous drop in capacity unit. This will indicate if Application Gateway is out of capacity.

Troubleshoot using metrics

There are other metrics that can indicate issues either at Application Gateway or the backend. We recommend evaluating alerts as per the table below.

METRIC	DESCRIPTION	USE CASE
Unhealthy Host Count	Number of backends that Application Gateway is unable to probe successfully.	Application Gateway instances are unable to connect to the backend. For example, the probe interval is 10 seconds and unhealthy host count threshold is 3 failed probes). A backend will turn unhealthy if Application Gateway instance isn't able to reach it for 30 seconds. Also depends on the configured timeout and interval in the custom probe.
Response Status (dimension 4xx and 5xx)	The HTTP response status returned to clients from Application Gateway. This status is usually same as the Backend Response Status , unless Application Gateway is unable to get a response from the backend or Application Gateway has an internal error in serving responses.	Issues with Application Gateway or the backend. Use this metric with Backend Response Status to identify whether Application Gateway or the backend is failing to serve requests.
Backend Response Status (dimension 4xx and 5xx)	The HTTP response status returned to Application Gateway from the backend.	Use to validate if the backend is successfully receiving requests and serving responses.
Backend Last Byte Response Time	Time interval between the start of a connection to backend server and receiving the last byte of the response body.	Increase in this latency implies that the backend is getting loaded and is taking longer to respond to requests. One way to resolve this issue is to scale up the backend.

METRIC	DESCRIPTION	USE CASE
Application Gateway Total Time	Time period from when Application Gateway receives the first byte of the HTTP request to when the last response byte has been sent to the client. This includes client RTT	Increase in this latency, without any accompanying application changes or access traffic pattern changes should be investigated. If this metric increases, monitor other the metrics and determine if they other metrics are also increasing, such as compute units, total throughput, or total request count.

Enable diagnostics on Application Gateway and web application firewall (WAF)

Diagnostic logs allow you to view firewall logs, performance logs, and access logs. Use these logs to manage and troubleshoot issues with Application Gateway instances.

Use Azure Monitor Network Insights

Azure Monitor Network Insights provides a comprehensive view of health and metrics for network resources, including Application Gateway. For additional details and supported capabilities for Application Gateway, see [Azure Monitor Network insights](#).

Use advanced monitoring metrics

Consider monitoring and setting alerts on metrics such as **Unhealthy host count**, and metrics that indicate the latency and the number of connections and requests. Notice the difference between connections and requests for Application Gateway frontend connections. One connection represents the TCP connection (sockets pair), while a request represents a resource request, that is a GET, PUT, POST, and so on. One connection can serve multiple requests (on Application Gateway v2, up to 100).

SNAT port limitations

SNAT port limitations are important for backend connections on the Application Gateway. There are separate factors that affect how Application Gateway reaches the SNAT port limit. For example, if the backend is a public IP, it will require its own SNAT port. In order to avoid SNAT port limitations, you can increase the number of instances per Application Gateway, scale out the backends to have more IPs, or move your backends into the same virtual network and use private IP addresses for the backends.

Requests per second (RPS) considerations

Requests per second (RPS) on the Application Gateway will be affected if the SNAT port limit is reached. For example, if Application Gateway has reached the SNAT port limit, then Application Gateway won't be able to open a new connection to the backend and the request will fail.

Match timeout settings with the backend application

Ensure you have configured the **IdleTimeout** settings to match the listener and traffic characteristics of the backend application. The default value is set to 4 minutes and can be configured to a maximum of 30. For more information, see [Load Balancer TCP Reset and Idle Timeout](#).

For workload considerations, see [Application Monitoring](#).

Monitoring Key Vault configuration issues through Azure Advisor

Azure Application Gateway checks for the renewed certificate version in the linked Key Vault at every 4-hour interval. If it is inaccessible due to any incorrectly modified Key Vault configurations, it logs that error and pushes a corresponding Advisor recommendation. You must configure Advisor alert to stay updated and fix such issues immediately to avoid any Control or Data plane related problems. To set an alert for this specific case, use the Recommendation Type as "Resolve Azure Key Vault issue for your Application Gateway".

Reliability

Here are some best practices to minimize failed instances.

In addition, we recommend that you review the [Principles of the reliability pillar](#).

Plan for rule updates

Plan enough time for updates before accessing Application Gateway or making further changes. For example, removing servers from backend pool might take some time because they have to drain existing connections.

Use health probes to detect backend unavailability

If Application Gateway is used to load balance incoming traffic over multiple backend instances, we recommend the use of health probes. These will ensure that traffic is not routed to backends that are unable to handle the traffic.

Review the impact of the interval and threshold settings on health probes

The health probe sends requests to the configured endpoint at a set *interval*. Also, there's a *threshold* of failed requests that will be tolerated before the backend is marked unhealthy. These numbers present a trade-off.

- Setting a higher interval puts a higher load on your service. Each Application Gateway instance sends its own health probes, so 100 instances every 30 seconds means 100 requests per 30 seconds.
- Setting a lower interval leaves more time before an outage is detected.
- Setting a low unhealthy threshold may mean that short, transient failures may take down a backend.
- Setting a high threshold it can take longer to take a backend out of rotation.

Verify downstream dependencies through health endpoints

Suppose each backend has its own dependencies to ensure failures are isolated. For example, an application hosted behind Application Gateway may have multiple backends, each connected to a different database (replica). When such a dependency fails, the application may be working but won't return valid results. For that reason, the health endpoint should ideally validate all dependencies. Keep in mind that if each call to the health endpoint has a direct dependency call, that database would receive 100 queries every 30 seconds instead of 1. To avoid this, the health endpoint should cache the state of the dependencies for a short period of time.

For more information, see these articles:

- [Health monitoring overview for Azure Application Gateway](#)
- [Azure Front Door - backend health monitoring](#)
- [Health probes to scale and provide HA for your service](#)

Security

Security is one of the most important aspects of any architecture. Application Gateway provides features to employ both the principle of least privilege and defense-in-defense. We recommend you also review the [Security design principles](#).

Restrictions of Network Security Groups (NSGs)

NSGs are supported on Application Gateway, but there are some restrictions. For instance, some communication with certain port ranges is prohibited. Make sure you understand the implications of those restrictions. For details, see [Network security groups](#).

User Defined Routes (UDR)-supported scenarios

Using User Defined Routes (UDR) on the Application Gateway subnet cause some issues. [Health status in the back-end](#) might be unknown. Application Gateway logs and metrics might not get generated. We recommend that you don't use UDRs on the Application Gateway subnet so that you can view the back-end health, logs, and metrics. If your organizations require to use UDR in the Application Gateway subnet, please ensure you review the supported scenarios. For details, see [Supported user-defined routes](#).

DNS lookups on App Gateway subnet

When the backend pool contains a resolvable FQDN, the DNS resolution is based on a private DNS zone or custom DNS server (if configured on the VNet), or it uses the default Azure-provided DNS.

Set up a TLS policy for enhanced security

Set up a [TLS policy](#) for extra security. Ensure you're using the latest TLS policy version (AppGwSslPolicy20170401S). This enforces TLS 1.2 and stronger ciphers.

Use AppGateway for TLS termination

There are advantages of using Application Gateway for TLS termination:

- Performance improves because requests going to different backends have to re-authenticate to each backend.
- Better utilization of backend servers because they don't have to perform TLS processing
- Intelligent routing by accessing the request content.
- Easier certificate management because the certificate only needs to be installed on Application Gateway.

Encrypting considerations

When re-encrypting backend traffic, ensure the backend server certificate contains both the root and intermediate Certificate Authorities (CAs). A TLS certificate of the backend server must be issued by a well-known CA. If the certificate was not issued by a trusted CA, the Application Gateway checks if the certificate of the issuing CA was issued by a trusted CA, and so on until either a trusted CA is found. Only then a secure connection is established. Otherwise, Application Gateway marks the backend as unhealthy.

Azure Key Vault for storing TLS certificates

[Application Gateway is integrated with Key Vault](#). This provides stronger security, easier separation of roles and responsibilities, support for managed certificates, and an easier certificate renewal and rotation process.

Enabling the Web Application Firewall (WAF)

When WAF is enabled, every request must be buffered by the Application Gateway until it fully arrives and checked if the request matches with any rule violation in its core rule set and then forward the packet to the backend instances. For large file uploads (30MB+ in size), this can result in a significant latency. Because Application Gateway capacity requirements are different with WAF, we do not recommend enabling WAF on Application Gateway without proper testing and validation.

Next steps

[Microsoft Azure Well-Architected Framework](#)

Azure Well-Architected Framework review of Azure Firewall

3/10/2022 • 13 minutes to read • [Edit Online](#)

This article provides architectural best practices for Azure Firewall. The guidance is based on the five pillars of architecture excellence: cost optimization, operational excellence, performance efficiency, reliability, and security.

Cost optimization

Review underutilized Azure Firewall instances, and identify and delete Azure Firewall deployments not in use. To identify Azure Firewall deployments not in use, start analyzing the Monitoring Metrics and User Defined Routes (UDRs) that are associated with subnets pointing to the Firewall's private IP. Then, combine that with additional validations, such as if the Azure Firewall has any Rules (Classic) for NAT, or Network and Application, or even if the DNS Proxy setting is configured to **Disabled**, as well as with internal documentation about your environment and deployments. See the details about monitoring logs and metrics at [Monitor Azure Firewall logs and metrics](#) and [SNAT port utilization](#).

Share the same Azure Firewall across multiple workloads and Azure Virtual Networks. Deploy a central Azure Firewall in the hub virtual network, and share the same Firewall across many spoke virtual networks that are connected to the same hub from the same region. Ensure that there is no unexpected cross-region traffic as part of the hub-spoke topology.

Stop Azure Firewall deployments that do not need to run for 24 hours. This could be the case for development environments that are used only during business hours. See more details at [Deallocate and allocate Azure Firewall](#).

Properly size the number of Public IPs that your firewall needs. Validate whether all the associated Public IPs are in use. If they are not in use, disassociate and delete them. Use IP Groups to reduce your management overhead. Evaluate SNAT ports utilization before you remove any IP Addresses. See the details about monitoring logs and metrics at [Monitor Azure Firewall logs and metrics](#) and [SNAT port utilization](#).

Use Azure Firewall Manager and its policies to reduce your operational costs, by increasing the efficiency and reducing your management overhead. Review your Firewall Manager policies, associations, and inheritance carefully. Policies are billed based on firewall associations. A policy with zero or one firewall association is free of charge. A policy with multiple firewall associations is billed at a fixed rate. See more details at [Pricing - Firewall Manager](#).

Review the differences between the two Azure Firewall SKUs. The Standard option is usually enough for east-west traffic, where Premium comes with the necessary additional features for north-south traffic, as well as the forced tunneling feature and many other features. See more information at [Azure Firewall Premium Preview features](#). Deploy mixed scenarios using the Standard and Premium options, according to your needs.

Operational excellence

General administration and governance

- Use Azure Firewall to govern:
 - Internet outbound traffic (VMs and services that access the internet)
 - Non-HTTP/S inbound traffic
 - East-west traffic filtering

- Use Azure Firewall Premium, if any of the following capabilities are required:
 - TLS inspection - Decrypts outbound traffic, processes the data, encrypts the data, and then sends it to the destination.
 - IDPS - A network intrusion detection and prevention system (IDPS) allows you to monitor network activities for malicious activity, log information about this activity, report it, and optionally attempt to block it.
 - URL filtering - Extends Azure Firewall's FQDN filtering capability to consider an entire URL. For example, the filtered URL might be www.contoso.com/a/c instead of www.contoso.com.
 - Web categories - Administrators can allow or deny user access to website categories, such as gambling websites, social media websites, and others.
 - See more details at [Azure Firewall Premium Preview features](#).
- Use Firewall Manager to deploy and manage multiple Azure Firewalls across Azure Virtual WAN hubs and hub-spoke based deployments.
- Create a global Azure Firewall policy to govern the security posture across the global network environment, and then assign it to all Azure Firewall instances. This allows for granular policies to meet the requirements of specific regions, by delegating incremental Azure Firewall policies to local security teams, via RBAC.
- Configure supported 3rd-party SaaS security providers within Firewall Manager, if you want to use such solutions to protect outbound connections.
- For existing deployments, migrate Azure Firewall rules to Azure Firewall Manager policies, and use Azure Firewall Manager to centrally manage your firewalls and policies.

Infrastructure provisioning and changes

- We recommend the Azure Firewall to be deployed in the hub VNet. Very specific scenarios might require additional Azure Firewall deployments in spoke virtual networks, but that is not common.
- Prefer using IP prefixes.
- Become familiar with the limits and limitations, especially SNAT ports. Do not exceed limits, and be aware of the limitations. See the Azure Firewall limits at [Azure subscription limits and quotas - Azure Resource Manager](#). Also, learn more about any existing usability limitations at [Azure Firewall FAQ](#).
- For concurrent deployments, make sure to use IP Groups, policies, and firewalls that do not have concurrent Put operations on them. Ensure all updates to the IP Groups and policies have an implicit firewall update that is run afterwards.
- Ensure a developer and test environment to validate firewall changes.
- A well-architected solution also involves considering the placement of your resources, to align with all functional and non-functional requirements. Azure Firewall, Application Gateway, and Load Balancers can be combined in multiple ways to achieve different goals. You can find scenarios with detailed recommendations, at [Firewall and Application Gateway for virtual networks](#).

Networking

An Azure Firewall is a dedicated deployment in your virtual network. Within your virtual network, a [dedicated subnet](#) is required for the Azure Firewall. Azure Firewall will provision more capacity as it scales. A /26 address space for its subnets ensures that the firewall has enough IP addresses available to accommodate the scaling. Azure Firewall does not need a subnet bigger than /26, and the Azure Firewall subnet name must be `AzureFirewallSubnet`.

- If you are considering using the forced tunneling feature, you will need an additional /26 address space for the Azure Firewall Management subnet, and you must name it `AzureFirewallManagementSubnet` (this is also a requirement).
- Azure Firewall always starts with two instances, it can scale up to 20 instances, and you cannot see those individual instances. You can only deploy a single Azure Firewall instance in each VNet.
- Azure Firewall must have direct Internet connectivity. If your `AzureFirewallSubnet` learns a default route to your on-premises network via BGP, then you must configure Azure Firewall in the forced tunneling mode. If

this is an existing Azure Firewall instance, which cannot be reconfigured in the forced tunneling mode, then we recommended that you create a UDR with a 0.0.0.0/0 route, with the **NextHopType** value set as **Internet**. Associate it with the **AzureFirewallSubnet** to maintain internet connectivity.

- When deploying a new Azure Firewall instance, if you enable the forced tunneling mode, you can set the Public IP Address to **None** to deploy a fully private data plane. However, the management plane still requires a public IP, for management purposes only. The internal traffic from Virtual Networks, and/or on-premises, will not use that public IP. See more about forced tunneling at [Azure Firewall forced tunneling](#).
- When you have multi-region Azure environments, remember that Azure Firewall is a regional service. Therefore, you'll likely have one instance per regional hub.

Monitoring

Monitoring capacity metrics

The following metrics can be used by the customer, as indicators of utilization of provisioned Azure Firewall capacity. Alerts can be set as needed by the customers, to get notifications once a threshold has been reached for any metric.

METRIC NAME	EXPLANATION
Application rules hit count	The number of times an application rule has been hit. Unit: count
Data processed	Sum of data traversing the firewall in a given time window. Unit: bytes
Firewall health state	Indicates the health of the firewall, based on SNAT port availability. Unit: percent This metric has two dimensions: - Status: Possible values are Healthy, Degraded, and Unhealthy. - Reason: Indicates the reason for the corresponding status of the firewall. If the SNAT ports are used > 95%, they are considered exhausted, and the health is 50% with status=Degraded and reason=SNAT port. The firewall keeps processing traffic, and the existing connections are not affected. However, new connections may not be established intermittently. If the SNAT ports are used < 95%, then the firewall is considered healthy, and the health is shown as 100%. If no SNAT ports usage is reported, then the health is shown as 0%.
Network rules hit count	The number of times a network rule has been hit. Unit: count

METRIC NAME	EXPLANATION
SNAT port utilization	<p>The percentage of SNAT ports that have been utilized by the firewall. Unit: percent</p> <p>When you add more public IP addresses to your firewall, then more SNAT ports are available. This reduces the SNAT ports utilization. Additionally, when the firewall scales out for different reasons (for example, CPU or throughput), then additional SNAT ports also become available. Effectively, a given percentage of SNAT ports utilization may go down without you adding any public IP addresses, just because the service scaled out. You can directly control the number of public IP addresses that are available, to increase the ports available on your firewall. But, you can't directly control firewall scaling.</p> <p>If your firewall is running into SNAT port exhaustion, you should add at least five public IP address. This increases the number of SNAT ports available. Another option is to associate a NAT Gateway with the Azure Firewall subnet, which can help you increase the ports up to +1M ports.</p>
Throughput	<p>Rate of data traversing the firewall, per second. Unit: bits per second</p>

Monitoring logs using Azure Firewall Workbook

Azure Firewall exposes a few other logs and metrics for troubleshooting that can be used as indicators of issues. We recommend evaluating alerts, as per the table below. Refer to [Monitor Azure Firewall logs and metrics](#).

METRIC NAME	EXPLANATION
Application rule log	Each new connection that matches one of your configured application rules will result in a log for the accepted/denied connection.
Network rule log	Each new connection that matches one of your configured network rules will result in a log for the accepted/denied connection.
DNS Proxy log	This log tracks DNS messages to a DNS server that is configured using a DNS proxy.

Diagnostics logs and policy analytics

- Diagnostic logs allow you to view Azure Firewall logs, performance logs, and access logs. You can use these logs in Azure to manage and troubleshoot your Azure Firewall instance.
- Policy analytics for Azure Firewall Manager allows you to start seeing rules and flows that match the rules and hit count for those rules. By watching what rule is in use and the traffic that's being matched, you can have full visibility of the traffic.

Performance efficiency

SNAT ports exhaustion

- If more than 512K ports are necessary, use a NAT gateway with Azure Firewall. To scale up that limit, you can have up to +1M ports when associating a NAT gateway to the Azure Firewall subnet. For more information, refer to [Scale SNAT ports with Azure NAT Gateway](#).

Auto scale and performance

- Azure Firewall uses auto scale. It can go up to 30 Gbps.
- Azure Firewall always starts with 2 instances. It scales up and down, based on CPU and the network throughput. After an auto scale, Azure Firewall ends up with either n-1 or n+1 instances.
- Scaling up happens if the threshold for CPU or throughput are greater than 60%, for more than five minutes.
- Scaling down happens if the threshold for CPU or throughput are under 60%, for more than 30 minutes. The scale-down process happens gracefully (deleting instances). The active connections on the deprovisioned instances are disconnected and switched over to other instances. For the majority of applications, this process does not cause any downtime, but applications should have some type of auto-reconnect capability. (The majority already has this capability.)
- If you're performing load tests, make sure to create initial traffic that is not part of your load tests, 20 minutes prior to the test. This is to allow the Azure Firewall instance to scale up its instances to the maximum. Use diagnostics settings to capture scale-up and scale-down events.
- Do not exceed 10k network rules, and make sure you use IP Groups. When creating network rules, remember that for each rule, Azure actually multiples **Ports x IP Addresses**, so if you have one rule with four IP address ranges and five ports, you will be actually consuming 20 network rules. Always try to summarize IP ranges.
- There are no restrictions for Application Rules.
- Add the Allow rules first, and then add the Deny rules to the lowest priority levels.

Reliability

- Azure Firewall provides different SLAs for when it is deployed in a single Availability Zone and for when it is deployed in multi-zones. For more information, see [SLA for Azure Firewall](#). For information about all Azure SLAs, see the [Azure service level agreements page](#).
- For workloads designed to be resistant to failures and to be fault-tolerant, remember to take into consideration that Azure Firewalls and Virtual Networks are regional resources.
- Closely monitor metrics, especially SNAT port utilization, firewall health state, and throughput.
- Avoid adding multiple individual IP addresses or IP address ranges to your network rules. Use super nets instead, or IP Groups when possible. Azure Firewall multiples **IPs x rules**, and that can make you reach the 10k recommended rules limit.

Security

- Understand rule processing logic:
 - Azure Firewall has NAT rules, network rules, and applications rules. The rules are processed according to the rule type. See more at [Azure Firewall rule processing logic](#) and [Azure Firewall Manager rule processing logic](#).
- Use FQDN filtering in network rules.
 - You can use FQDNs in network rules, based on DNS resolution in Azure Firewall and Firewall policy. This capability allows you to filter outbound traffic with any TCP/UDP protocol (including NTP, SSH, RDP, and more). You must enable the DNS proxy to use FQDNs in your network rules. See how it works at [Azure Firewall FQDN filtering in network rules](#).
- If you're filtering inbound Internet traffic with Azure Firewall policy DNAT, for security reasons, then the recommended approach is to add a specific Internet source, to allow DNAT access to the network and to avoid using wildcards.
- Use Azure Firewall to secure private endpoints (the virtual WAN scenario). See more at [Secure traffic destined to private endpoints in Azure Virtual WAN](#).
- Configure threat intelligence:

- Threat-intelligence-based filtering can be configured for your Azure Firewall policy to alert and deny traffic from and to known malicious IP addresses and domains. See more at [Azure Firewall threat intelligence configuration](#).
- Use Azure Firewall Manager:
 - Azure Firewall Manager is a security management service that provides a central security policy and route management for cloud-based security perimeters. It includes the following features:
 - Central Azure Firewall deployment and configuration.
 - Hierarchical policies (global and local).
 - Integrated with third-party security-as-a-service for advanced security.
 - Centralized route management.
 - Understand how Policies are applied, at [Azure Firewall Manager policy overview](#).
 - Use Azure Firewall policy to define a rule hierarchy. See [Use Azure Firewall policy to define a rule hierarchy](#).
- Use Azure Firewall Premium:
 - Azure Firewall Premium is a next-generation firewall, with capabilities that are required for highly sensitive and regulated environments. It includes the following features:
 - TLS inspection - Decrypts outbound traffic, processes the data, encrypts the data, and then sends it to the destination.
 - IDPS - A network intrusion detection and prevention system (IDPS) allows you to monitor network activities for malicious activity, log information about this activity, report it, and optionally attempt to block it.
 - URL filtering - Extends Azure Firewall's FQDN filtering capability to consider an entire URL. For example, the filtered URL might be www.contoso.com/a/c instead of www.contoso.com.
 - Web categories - Administrators can allow or deny user access to website categories, such as gambling websites, social media websites, and others.
 - See more at [Azure Firewall Premium Preview features](#).
- Deploy a security partner provider:
 - Security partner providers, in Azure Firewall Manager, allow you to use your familiar, best-in-breed, third-party security as a service (SECaaS) offering to protect Internet access for your users.
 - With a quick configuration, you can secure a hub with a supported security partner. You can route and filter Internet traffic from your Virtual Networks (VNets) or branch locations within a region. You can do this with automated route management, without setting up and managing user-defined routes (UDRs).
 - The current supported security partners are Zscaler, Check Point, and iboss.
 - See more at [Deploy an Azure Firewall Manager security partner provider](#).

Next steps

- See the [Microsoft Azure Well-Architected Framework](#).
- [What is Azure Firewall?](#)

Related resources

- [Azure Firewall architecture overview](#)
- [Azure Well-Architected Framework review of Azure Application Gateway](#)
- [Firewall and Application Gateway for virtual networks](#)
- [Choose between virtual network peering and VPN gateways](#)
- [Hub-spoke network topology in Azure](#)
- [Security considerations for highly sensitive IaaS apps in Azure](#)

Storage architecture design

3/10/2022 • 2 minutes to read • [Edit Online](#)

The Azure Storage platform is the Microsoft cloud storage solution for modern data storage scenarios.

The Azure Storage platform includes the following data services:

- [Azure Blob Storage](#): A massively scalable object store for text and binary data. Also includes support for big data analytics through Azure Data Lake Storage Gen2.
- [Azure Files](#): Managed file shares for cloud or on-premises deployments.
- [Azure Queue Storage](#): A messaging store for reliable messaging between application components.
- [Azure Table Storage](#): A NoSQL store for schemaless storage of structured data.
- [Azure Disk Storage](#): Block-level storage volumes for Azure VMs.

Introduction to storage on Azure

If you're new to storage on Azure, the best way to learn more is with [Microsoft Learn](#), a free online training platform. Microsoft Learn provides interactive learning for Microsoft products and more. Check out the [Store data in Azure](#) learning path.

Path to production

- Choose the storage approach that best meets your needs and then create an account. For more information, see [Storage account overview](#).
- Be sure you understand security and reliability. See these articles:
 - [Azure Storage encryption for data at rest](#)
 - [Use private endpoints - Azure Storage](#)
 - [Data redundancy - Azure Storage](#)
 - [Disaster recovery and storage account failover - Azure Storage](#)
- For information about migrating existing data, see the [Azure Storage migration guide](#).

Best practices

Depending on the storage technology you use, see the following best practices resources:

- [Performance and scalability checklist for Blob Storage](#)
- [Best practices for using Azure Data Lake Storage Gen2](#)
- [Planning for an Azure Files deployment](#)
- [Performance and scalability checklist for Queue Storage](#)
- [Azure Storage table design patterns](#)

Blob Storage

See the following guides for information about Blob Storage:

- [Authorize access to blobs using Azure Active Directory](#)
- [Security recommendations for Blob Storage](#)

Azure Data Lake Storage

See the following guides for information about Data Lake Storage:

- [Best practices for using Azure Data Lake Storage Gen2](#)
- [Azure Policy Regulatory Compliance controls for Azure Data Lake Storage Gen1](#)

Azure Files

See the following guides for information about Azure Files:

- [Planning for an Azure Files deployment](#)
- [Overview of Azure Files identity-based authentication options for SMB access](#)
- [Disaster recovery and storage account failover](#)
- [About Azure file share backup](#)

Queue Storage

See the following guides for information about Queue Storage:

- [Authorize access to queues using Azure Active Directory](#)
- [Performance and scalability checklist for Queue Storage](#)

Table Storage

See the following guides for information about Table Storage:

- [Authorize access to tables using Azure Active Directory \(preview\)](#)
- [Performance and scalability checklist for Table storage](#)
- [Design scalable and performant tables](#)
- [Design for querying](#)

Azure Disk Storage

See the following guides for information about Azure managed disks:

- [Server-side encryption of Azure Disk Storage](#)
- [Azure Disk Encryption for Windows VMs](#)
- [Azure premium storage: design for high performance](#)
- [Scalability and performance targets for VM disks](#)

Stay current with storage

Get the [latest updates on Azure Storage products and features](#).

Additional resources

To plan for your storage needs, see [Review your storage options](#).

Example solutions

Here are a few sample implementations of storage on Azure:

- [Using Azure file shares in a hybrid environment](#)
- [Azure files accessed on-premises and secured by AD DS](#)
- [Enterprise file shares with disaster recovery](#)

- Hybrid file services
- Optimized storage with logical data classification
- Medical data storage solutions
- HPC media rendering

See more storage examples in the [Azure Architecture Center](#).

AWS or Google Cloud professionals

These articles provide service mapping and comparison between Azure and other cloud services. They can help you ramp up quickly on Azure.

- [Compare AWS and Azure Storage services](#)
- [Google Cloud to Azure services comparison - Storage](#)

Azure AD join for Azure Virtual Desktop

3/10/2022 • 7 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) provides many benefits for organizations, such as modern authentication protocols, single sign-on (SSO), and support for [FSLogix](#) user profiles. Azure Virtual Desktop virtual machine (VM) session hosts can join directly to Azure AD. Joining directly to Azure AD removes the previous need to use Active Directory Domain Services (AD DS) domain controllers.

Originally, Azure Virtual Desktop domain join needed both Azure AD and AD DS domain controllers. Traditional Windows Server AD DS domain controllers were on-premises machines, Azure VMs, or both. Azure Virtual Desktop accessed the controllers over a site-to-site virtual private network (VPN) or Azure ExpressRoute. Alternatively, [Azure Active Directory Domain Services](#) platform-as-a-service (PaaS) provided AD DS in Azure and supported trust relationships to existing on-premises AD DS. Users had to sign in to both Azure AD and AD DS.

Other services that Azure Virtual Desktop hosts consume, such as applications and Server Message Block (SMB) storage, might still require AD DS. But Azure Virtual Desktop itself no longer requires AD DS. Removing this requirement reduces cost and complexity.

Azure AD domain join for Azure Virtual Desktop provides a modern approach for smartcards, FIDO2, authentication protocols like Windows Hello for Business, and future capabilities. Azure AD domain join also opens up the possibility of decommissioning Active Directory, since Azure Virtual Directory host pools no longer require Active Directory.

This article describes how to configure Azure AD domain join for Azure Virtual Desktop, along with some troubleshooting tips. For most Windows Azure Virtual Desktop clients, the Azure AD join configuration consists of two steps, deploying the host pool and enabling user access. For non-Windows Azure Virtual Desktop clients and other cases that need further configuration, see [Protocol and client options](#).

Prerequisites

There are a few limitations for Azure Virtual Desktop Azure AD domain join:

- Azure AD join is only supported on Azure Virtual Desktop for Azure Resource Manager. Azure Virtual Desktop Classic isn't supported.
- Only personal host pools are currently supported. This limitation isn't in multisession pooled host pools, but in Azure Files. Azure Files currently doesn't support Azure AD as a [Kerberos](#) realm, only Active Directory. This lack of Kerberos support prevents FSLogix from working. FSLogix is the technology that manages roaming user profiles in a pooled host pool scenario.
- The session hosts must be Windows 10 Enterprise version 2004 or later.

Step 1: Deploy an Azure AD join host pool

To deploy an Azure AD host pool, follow the instructions in [Create a host pool](#). On the [Create a host pool](#) screen, on the **Virtual Machines** tab, under **Domain to join**, select **Azure Active Directory**.

Domain to join

Select which directory you would like to join

Active Directory

Active Directory

AD domain join UPN * ⓘ

Azure Active Directory

Selecting **Azure Active Directory** presents the option to enroll the VMs with Intune. Select **Yes** if you want to enroll the VM with Intune.

Intune can apply policies, distribute software, and help you manage VMs. For more information about Intune as part of Microsoft Endpoint Manager, see [Getting started with Microsoft Endpoint Manager](#).

Domain to join

Select which directory you would like to join

Azure Active Directory

Enroll VM with Intune ⓘ

Yes No

In the deployment, a new extension called **AADLoginForWindows** creates the Azure AD join and the Intune enrollment if selected.

Your deployment is complete

(1) Deployment name: vmCreation-linkedTemplate-2c966c18-9411-4e... Start time: 5/6/2021, 3:23:16 PM
Subscription: Tom's Microsoft Subscription - XenithIT AAD Correlation ID: bf043184-e562-4e8b-9419-2d8
Resource group: AADJ1

Deployment details ([Download](#))

Resource	Type	Status
<input checked="" type="checkbox"/> AADJ1-1/AADLoginForWindowsWithIntune	Microsoft.Compute/virtualMachines/extensions	OK
<input checked="" type="checkbox"/> AADJ1-0/AADLoginForWindowsWithIntune	Microsoft.Compute/virtualMachines/extensions	OK

You can also add session hosts to an existing host pool and have them Azure AD joined and Intune enrolled.

After you create the host pool VMs, you can see the VMs in **Azure AD > Devices**.

Name	Enabled	OS	Version	Join Type	Owner	User name	MDM
<input type="checkbox"/> AADJ2-1	<input checked="" type="radio"/> Yes	Windows	10.0.19042.928	Azure AD joined	Chris	Chris@xenithit.com	Microsoft Intune
<input type="checkbox"/> Client	<input checked="" type="radio"/> Yes	Windows	10.0.19042.928	Azure AD joined	Tom	Tom@xenithit.com	Microsoft Intune
<input type="checkbox"/> AADJ1-0	<input checked="" type="radio"/> Yes	Windows	10.0.21390.2025	Azure AD joined	None	None	Microsoft Intune

To confirm Azure AD registrations, go to **Azure Active Directory > Devices > Audit Logs** and look for **Register device**.

6/25/2021, 2:25:05 PM	Device Registration Service	Device	Register device	Success
6/28/2021, 3:48:27 PM	Device Registration Service	Device	Register device	Success
6/22/2021, 3:35:10 PM	Device Registration Service	Device	Register device	Success
6/25/2021, 2:14:49 PM	Device Registration Service	Device	Register device	Success

The VMs also appear in the [MEM portal](#), in the **Devices** section.

Device name ↑↓	Managed by ↑↓	Ownership ↑↓	Compliance ↑↓	OS	OS version ↑↓
AADJ1-0	Intune	Corporate	<input checked="" type="radio"/> Compliant	Windows	10.0.21390.2025
AADJ1-1	Intune	Corporate	<input checked="" type="radio"/> Compliant	Windows	10.0.22000.51
AADJ2-0	Intune	Corporate	<input checked="" type="radio"/> Compliant	Windows	10.0.19042.928
AADJ2-1	Intune	Corporate	<input checked="" type="radio"/> Compliant	Windows	10.0.19042.928

If a VM doesn't appear or you want to confirm enrollment, sign in to the VM locally and at a command prompt,

run the following command:

```
dsregcmd /status
```

The output shows the VM's Azure AD join status.

```
C:\>dsregcmd /status
+-----
| Device State
+-----

    AzureAdJoined : YES
    EnterpriseJoined : NO
    DomainJoined : NO
    Device Name : test-0
```

On the local client, the Azure AD registration logs are in Event Viewer at **Applications and Services Logs > Microsoft > Windows > User Device Registration > Admin**.

NOTE

With the previous, AD DS scenario, you could manually deploy session host VMs in a separate subscription connected to a different Azure AD if necessary. The VMs had no dependency on Azure AD. The VMs only needed network line of sight to an AD DS domain controller in a domain that synchronized user objects to the Azure Virtual Desktops' Azure AD.

Azure AD join doesn't support this scenario. The host VMs automatically join to the Azure AD of the subscription that deploys the VMs. The deployment inherits that Azure AD as an identity provider, and uses the user identities that the Azure AD holds. There's no way to specify a different Azure AD for the host VMs. So be sure to create the VMs in the same subscription as all the other Azure Virtual Desktop objects. The VMs also automatically enroll into the Intune tenant associated with the Azure AD.

Step 2: Enable user access

In the next step, you enable sign-in access to the VMs. These VMs are Azure objects, and the authentication mechanism is Azure AD. You manage user sign-in permission through Azure role-based access control (RBAC).

In Azure Virtual Desktop, users must be in the Azure Virtual Desktop **Desktop application group** to sign in to the VMs. For Azure AD join, the same users and groups that are in the Desktop application group must also be added to the **Virtual Machine User Login** RBAC role. This role isn't a **Azure Virtual Desktop role**, but an Azure role with the **Log in to Virtual Machine** DataAction permission.

Virtual Machine User Login

BuiltInRole

Permissions JSON Assignments

Description: View Virtual Machines in the portal and login as a regular user.

Search permissions Type : All

Actions DataActions

Showing 1 of 1 permissions

Type	Permissions	Description
✓ Microsoft.Compute	Log in to Virtual Machine	Log in to a virtual machine as a regular user

Choose the scope for this role.

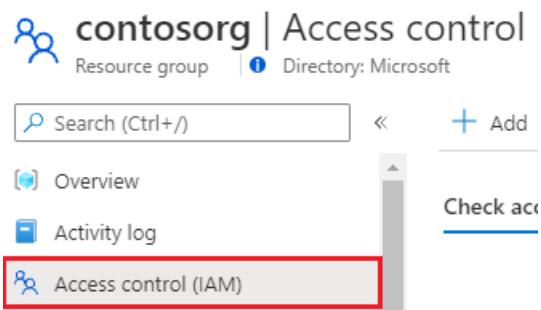
- Assigning the role at the **VM level** means you have to assign the role for each VM you add.

- Assigning the role at the **resource group level** means the role automatically applies to all VMs in that resource group.
- Assigning the role at the **Subscription level** means users can sign in to all VMs in the subscription.

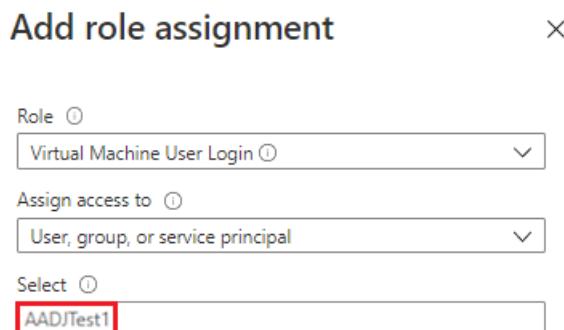
Setting the role once at the resource group level might be the best option. This approach prevents having to assign the role for every VM, but avoids assigning it at the top level of the subscription.

To assign the **Virtual Machine User Login** role:

1. In the Azure portal, go to your chosen scope, for example the resource group, and select **Access control (IAM)**.



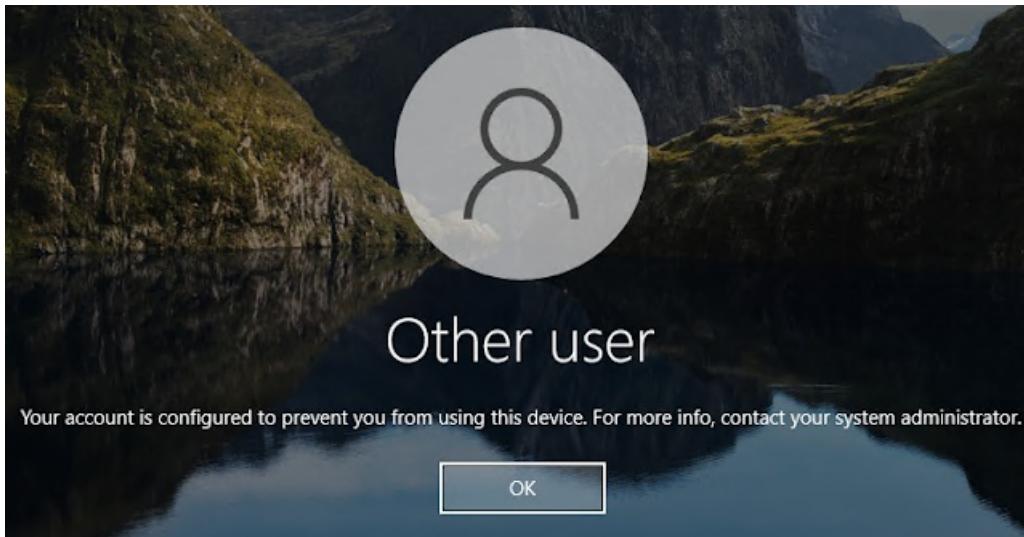
2. At the top of the screen, select + **Add > Add role assignment**.
3. Under **Role**, select **Virtual Machine User Login**, and under **Select**, select the same user group that's assigned to the Desktop Application Group.



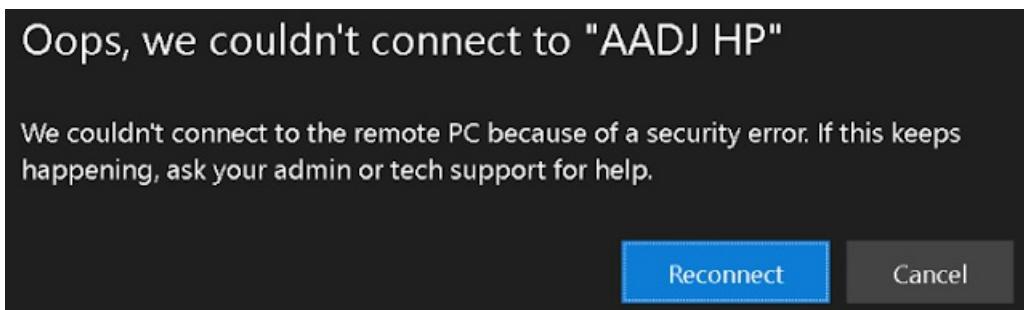
The user group now appears under **Virtual Machine User Login**.

Virtual Machine User Login		
<input type="checkbox"/>	AADJTest1 AADJTest1@xenithit.com	User
		Virtual Machine User Login ⓘ

If you don't assign this role, users get an error message when they try to sign in via the Windows client.



Web client users get a different-looking error.



Local Admin access

To give a user local administrative access to the VM, also add the user to the **Virtual Machine Administrator Login** role. This role has a **Log in to Virtual Machine as administrator** DataAction permission that enables administrative access.

Virtual Machine Administrator Login
BuiltInRole

Permissions JSON Assignments

Description: View Virtual Machines in the portal and login as administrator

Search permissions Type : All

Actions DataActions

Showing 2 of 2 permissions

Type	Permissions	Description
Other	Log in to Virtual Machine	Log in to a virtual machine as a regular user
Other	Log in to Virtual Machine as administrator	Log in to a virtual machine with Windows administrator or Linux root user privileges

Protocol and client options

By default, host pool access only works from the [Windows Azure Virtual Desktop client](#). To access host pool VMs, your local computer must be:

- Azure AD-joined or hybrid Azure AD-joined to the same Azure AD tenant as the session host.
- Running Windows 10 version 2004 or later, and also Azure AD-registered to the same Azure AD tenant as the session host.

Host pool access uses the Public Key User to User (PKU2U) protocol for authentication. To sign in to the VM, the session host and the local computer must have the PKU2U protocol enabled. For Windows 10 version 2004 or later machines, if the PKU2U protocol is disabled, enable it in the Windows registry as follows:

1. Navigate to HKLM\SYSTEM\CurrentControlSet\Control\Lsa\pku2u.

2. Set AllowOnlineID to 1.

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\pku2u			
	Name	Type	Data
	AllowOnlineID	REG_DWORD	0x00000001 (1)

If your client computers use Group Policy, also enable the Group Policy Option:

1. Navigate to Computer Configuration\Policies\Windows Settings\Security Settings\Local Policies\Security Options.
2. Under Policy, set Network security: Allow PKU2U authentication requests to this computer to use online identities to Enabled.

The screenshot shows the Group Policy Editor interface. On the left, there's a navigation tree under 'Computer Configuration' with various policy categories like Software Settings, Windows Settings, and Security Settings. In the main pane, a specific policy named 'Network security: Allow PKU2U authentication requests to this computer to use online identities.' is selected. A detailed configuration window for this policy is open, showing the setting 'Define this policy setting:' with the radio button 'Enabled' selected. To the right of the policy list, there's a vertical column of status indicators for other policies, all of which are currently 'Not Defined' except for the one shown which is 'Enabled'.

If you're using other Azure Virtual Desktop clients, such as Mac, iOS, Android, web, the Store client, or pre-version 2004 Windows 10, enable the **RDSTLS protocol**. Enable this protocol by adding a new **custom RDP Property** to the host pool, `targetIsaadjoined:i:1`. Azure Virtual Desktop then uses this protocol instead of PKU2U.

The screenshot shows the Azure portal's 'Host pools' blade. On the left, there's a sidebar with options like 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. The 'RDP Properties' section is highlighted. In the main area, a text input field contains the value '`targetIsaadjoined:i:1`', which is highlighted with a red box. Below the input field, a note says: 'The characters \; ; must be escaped with a backslash character \' from properties values. e.g. 'C' should be 'C\'.'

Now you have an Azure Virtual Desktop host pool where the session hosts are joined only to Azure AD. You're a step closer to modern management for your Azure Virtual Desktop estate.

Next steps

- [Azure Virtual Desktop documentation](#)
- [Deploy Azure AD-joined virtual machines in Azure Virtual Desktop](#)

Related resources

- [Azure Virtual Desktop for the enterprise](#)
- [Integrate on-premises AD domains with Azure AD](#)

Web architecture design

3/10/2022 • 3 minutes to read • [Edit Online](#)

Today's web apps are expected to be available all day, every day from anywhere in the world, and usable from virtually any device or screen size. Web applications must be secure, flexible, and scalable to meet spikes in demand.

This article provides an overview of Azure web app technologies, guidance, solution ideas, and reference architectures.

Azure provides a wide range of tools and capabilities for creating, hosting, and monitoring web apps. These are just some of the key web app services available in Azure:

- [Azure App Service](#) enables you to easily create enterprise-ready web and mobile apps for any platform or device and deploy them on a scalable cloud infrastructure.
- [Azure Web Application Firewall](#) provides powerful protection for web apps.
- [Azure Monitor](#) provides full observability into your applications, infrastructure, and network. Monitor includes [Application Insights](#), which provides application performance management and monitoring for live web apps.
- [Azure SignalR Service](#) enables you to easily add real-time web functionalities.
- [Static Web Apps](#) provides streamlined full-stack development, from source code to global high availability.
- [Web App for Containers](#) enables you to run containerized web apps on Windows and Linux.

Introduction to web apps on Azure

If you're new to creating and hosting web apps on Azure, the best way to learn more is with [Microsoft Learn](#), a free online training platform. Microsoft Learn provides interactive training for Microsoft products and more.

These are a few good starting points to consider:

- Learning path:
 - [Create Azure App Service web apps](#)
- Individual modules:
 - [Deploy and run a containerized web app with Azure App Service](#)
 - [Azure Static Web Apps](#)

Path to production

Consider these patterns, guidelines, and architectures as you plan and implement your deployment:

- [Basic web application](#)
- [Common web application architectures](#)
- [Design principles for Azure applications](#)
- [Design and implementation patterns - Cloud Design Patterns](#)
- [Enterprise deployment using App Services Environment](#)
- [High availability enterprise deployment using App Services Environment](#)

Best practices

For a good overview, see [Characteristics of modern web applications](#).

For information specific to Azure App Service, see these resources:

- [Azure App Service and operational excellence](#)
- [App Service deployment best practices](#)
- [Security recommendations for App Service](#)
- [Azure security baseline for App Service](#)

Web app architectures

The following sections, organized by category, provide links to sample web app architectures.

E-commerce

- [E-commerce front end](#)
- [Intelligent product search engine for e-commerce](#)
- [Scalable order processing](#)
- [E-commerce website running in secured App Service Environment](#)
- [Scalable e-commerce web app](#)
- [Scalable Episerver marketing website](#)
- [Scalable Sitecore marketing website](#)
- [Simple digital marketing website](#)

Healthcare

- [Clinical insights with Microsoft Cloud for Healthcare](#)
- [Consumer health portal on Azure](#)
- [Virtual health on Microsoft Cloud for Healthcare](#)

Modernization

- [Choose between traditional web apps and single-page apps](#)
- [ASP.NET architectural principles](#)
- [Common client-side web technologies](#)
- [Development process for Azure](#)
- [Azure hosting recommendations for ASP.NET Core web apps](#)

Multi-tier apps

- [Multi-tier app service with private endpoint](#)
- [Multi-tier app service with service endpoint](#)
- [Multi-tier web application built for HA/DR](#)

Multi-region apps

- [Highly available multi-region web application](#)
- [Multi-region web app with private connectivity to database](#)

Scalability

- [Scalable and secure WordPress on Azure](#)
- [Scalable cloud applications and site reliability engineering \(SRE\)](#)
- [Scalable web application](#)
- [Scalable Umbraco CMS web app](#)
- [Scalable web apps with Azure Redis Cache](#)

Security

- [Improved-security access to multitenant web apps from an on-premises network](#)

- Protect APIs with Application Gateway and API Management

SharePoint

- Highly available SharePoint farm
- Hybrid SharePoint farm with Microsoft 365

Stay current with web development

Get the latest [updates on Azure web app products and features](#).

Additional resources

Example solutions

Here are some additional implementations to consider:

- Simple branded website
- Build web and mobile applications
- Eventual consistency between multiple Power Apps instances
- App Service networking features
- IaaS: Web application with relational database
- Migrate a web app using Azure APIM
- Sharing location in real time using low-cost serverless Azure services
- Serverless web application
- Web application monitoring on Azure
- Web app private connectivity to Azure SQL Database
- Dynamics Business Central as a service on Azure
- Real-time presence with Microsoft 365, Azure, and Power Platform

AWS or Google Cloud professionals

- AWS to Azure services comparison - Web applications
- Google Cloud to Azure services comparison - Application services

Ten design principles for Azure applications

3/10/2022 • 2 minutes to read • [Edit Online](#)

Follow these design principles to make your application more scalable, resilient, and manageable.

Design for self healing. In a distributed system, failures happen. Design your application to be self healing when failures occur.

Make all things redundant. Build redundancy into your application, to avoid having single points of failure.

Minimize coordination. Minimize coordination between application services to achieve scalability.

Design to scale out. Design your application so that it can scale horizontally, adding or removing new instances as demand requires.

Partition around limits. Use partitioning to work around database, network, and compute limits.

Design for operations. Design your application so that the operations team has the tools they need.

Use managed services. When possible, use platform as a service (PaaS) rather than infrastructure as a service (IaaS).

Use the best data store for the job. Pick the storage technology that is the best fit for your data and how it will be used.

Design for evolution. All successful applications change over time. An evolutionary design is key for continuous innovation.

Build for the needs of business. Every design decision must be justified by a business requirement.

Design and implementation patterns

3/10/2022 • 2 minutes to read • [Edit Online](#)

Good design encompasses factors such as consistency and coherence in component design and deployment, maintainability to simplify administration and development, and reusability to allow components and subsystems to be used in other applications and in other scenarios. Decisions made during the design and implementation phase have a huge impact on the quality and the total cost of ownership of cloud hosted applications and services.

PATTERN	SUMMARY
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.
Anti-Corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.
Backends for Frontends	Create separate backend services to be consumed by specific frontend applications or interfaces.
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.
Compute Resource Consolidation	Consolidate multiple tasks or operations into a single computational unit
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.
Gateway Routing	Route requests to multiple services using a single endpoint.
Leader Election	Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the other instances.
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.

PATTERN	SUMMARY
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

