M.Sc. (IT)

SEMESTER - III (CBCS)

CLOUD APPLICATION DEVELOPMENT

SUBJECT CODE : PSIT302C

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai,

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai,

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai,

Programme Co-ordinator : **Prof. Mandar Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai, Mumbai

Course Co-ordinator **:** **Ms. Preeti Bharanuke**
Asst. Professor, M.Sc. I.T.
IDOL, University of Mumbai, Mumbai

Editor **:** **Prof. Hiren Dand**
Head of Department, IT,
M.C.C. College, Mulund, Mumbai

Course Writers **:** **Ms. Hema Sachin Darne**
Dr. Moonje Institute of Management and
Computer Studies, Nashik

: **Dr. Gautam Murlidhar Borkar**
Ramrao Adik Institute of Technology,

: **Mr. Vikram Ramlakhan Patalbansi**
Late Bhausaheb Hiray S.S. Trust's Institute of
Computer Application Bandra East, Mumbai

: **Dr. Padmaleela Dhamraju**
Sies College, Nerul

: **Miss Sharmila Junnarkar - Pote**
Bharat College of Arts and Commerce

October 2021, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai

CONTENTS

M. Sc (Information Technology)
Semester – III
Cloud Application Development

Course Objectives:

- To develop and deploy Microservices for cloud
- To understand Kubernetes and deploy applications on Azure Kubernetes Service
- To understand DevOps for Azure
- To follow the DevOps practices for software development
- To build APIs for Azure and AWS

Unit I :

Implementing Microservices: Client to microservices communication, Interservice communication, data considerations, security, monitoring, microservices hosting platform options.

Azure Service Fabric: Introduction, core concepts, supported programming models, service fabric clusters, develop and deploy applications of service fabric.

Monitoring Azure Service Fabric Clusters: Azure application, resource manager template, Adding Application Monitoring to a Stateless Service Using Application Insights, Cluster monitoring, Infrastructure monitoring.

Unit II :

Azure Kubernetes Service (AKS): Introduction to kubernetes and AKS, AKS development tools, Deploy applications on AKS.

Monitoring AKS: Monitoring, Azure monitor and analytics, monitoring AKS clusters, native kubernetes dashboard, Prometheus and Grafana.

Securing Microservices: Authentication in microservices, Implenting security using API gateway pattern, Creating application using Ocrlot and securing APIs with Azure AD.

Database Design for Microservices: Data stores, monolithic approach, Microservices approach, harnessing cloud computing, dataase options on MS Azure, overcoming application development challenges.

Building Microservices on Azure Stack: Azure stack, Offering IaaS, PaaS on-premises simplified, SaaS on Azure Stack.

Unit III :

.NET DevOps for Azure: DevOps introduction, Problem and solution.

Professional Grade DevOps Environment: The state of DevOps, professional grade DevOps vision, DevOps architecture, tools for professional DevOps environment, DevOps centered application.

Tracking work: Process template, Types of work items, Customizing the process, Working with the process.

Tracking code: Number of repositories, Git repository, structure, branching pattern, Azure repos configuration, Git and Azure.

Unit IV :

Building the code: Structure of build, using builds with .NET core and Azure pipelines, Validating the code: Strategy for defect detection, Implementing defect detection.

Release candidate creation: Designing release candidate architecture, Azure artifacts workflow for release candidates, Deploying the release: Designing deployment pipeline, Implementing deployment in Azure pipelines.

Operating and monitoring release: Principles, Architectures for observability, Jumpstarting observability.

Unit V :

Introduction to APIs: Introduction, API economy, APIs in public sector.

API Strategy and Architecture: API Strategy, API value chain, API architecture, API management.

API Development: Considerations, Standards, kick-start API development, team orientation.

API Gateways: API Gateways in public cloud, Azure API management, AWS API gateway.

API Security: Request-based security, Authentication and authorization.

Sr. No.	Title	Author/s	Publisher	Year
1	Building Microservices Applications on Microsoft Azure- Designing, Developing, Deploying, and Monitoring	Harsh Chawla Hemant Kathuria	Apress	2019
2	.NET DevOps for Azure A Developer's Guide to DevOps Architecture the Right Way	Jeffrey Palermo	Apress	2019
3	Practical API Architecture and Development with Azure and AWS - Design and Implementation of APIs for the Cloud	Thurupathan Vijayakumar	Apress	2018

❖❖❖❖

Unit I

1

IMPLEMENTING MICROSERVICES

Unit Structure :

1.0 OBJECTIVES

Most companies traditionally start by their infrastructures equaling a single monolith or several rigid, commonly dependent monolithic applications. Those monolith applications give multiple functions for

which the complete programming is prepared in an interconnected and regular portion of the application code.

If the requirement occurs, detangling the application code is hard as the code for these functions is joined as one. Any tries to edit or append even a single character in the monolith application can interrupt the complete code for the application, which provides development a long, slow, and costly method.

This is why microservices architecture is preferred by many companies those times as the system cuts the big monolith applications into autonomous, smaller, and loosely coupled elements. Those little elements are created to enhance effectiveness for separate tasks the communication with other components occurs within the simple, commonly available APIs.

This microservices architectural model is considered to be predominantly perfect when there is a need for help for different platforms and appliances, such as desktop, mobile, web, Internet of Things, etc. This model is perfect when applications are big as it is simpler to control when the architecture is broken down into tinier, independent modules that operate in unity.

In microservices architecture, as each independent element operates on a different method, it gives the IT teams a more decentralized methodology in producing great software and also allows them to maintain the separate component to be used, reproduced, rearranged, and controlled independently.

1.1 INTRODUCTION

The fundamental intention behind microservices is that some kind of applications become simpler to create and support during all are split down into smaller, composable parts that operate collectively. Every component is continuously improved and prepared, and the application is then just the sum of its electing components. This is in reverse to a regular, "monolithic" application which is all developed all in one piece.

Applications developed as a collection of modular components are more obvious to understand, more comfortable to test, and most importantly more informal to control across the life of the application. It enables corporations to gain much more formidable activity and be ready to greatly increase the time it takes to become effective changes to production. This method has shown to be better, particularly for big industry applications which are produced by teams of geographically and culturally different developers.

There are other advantages:

- **Developer independence**: Small organizations operate in correspondence and can repeat quicker than big organizations.
- **Isolation and resilience**: If a component fails, you turn up another while and the remainder of the application continues to function.
- **Scalability**: Smaller components take up fewer sources and can be scaled to match the growing need of that component only.
- **Lifecycle automation**: Unique components are more comfortable to fit into constant transmission pipelines and complicated deployment situations not possible with monoliths.
- B**usiness Relationship**: Microservice architectures are divided on business domain boundaries, increasing independence, and understanding over the organization.

The common explanation of microservices depends on all microservice providing an API endpoint, a stateless REST API that can be obtained over HTTP(S) just similar to a standardweb page. This process for obtaining microservices gives them simply for developers to use as they only need tools and methods many developers are already familiar with.

1.2 AN OVERVIEW

The idea of dividing applications into smaller pieces is nothing new; other programming paradigms address this same idea, such as Service Oriented Architecture (SOA). But, current technology improvements linked with an increasing expectation of unified "digital experiences" have provided rise to a new breed of development tools and techniques used to meet the needs of modern business applications. Microservices depend on the technology to help this thought, but an organization has the experience and structures in position for expansion teams capable of approving this model. Microservices are a band of a larger group in IT departments towards a Dev Ops culture. The development and services teams operate collectively to help an application over its lifecycle or even constant discharge cycle rather than a more conventional one.

Why is open source important for microservices?

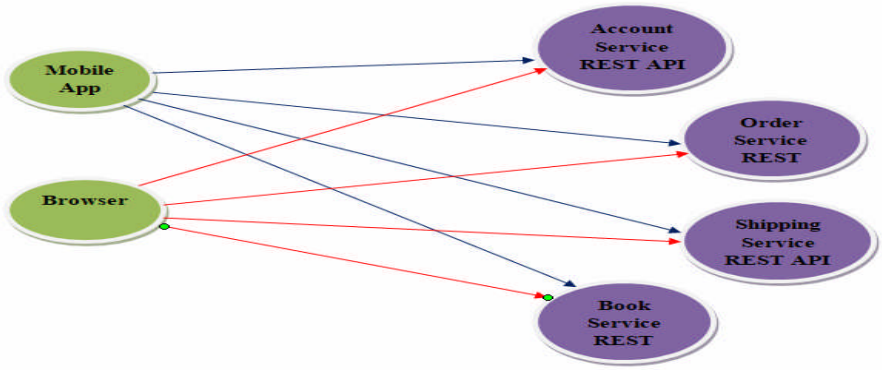
When you compose your applications from the beginning up to be modular and composable, it permits you to utilize drop-in components in many areas where in the past you may have needed established solutions, either because of the license for the components or functional requirements. Many application components can be off-the-shelf open-source tools, multiple open-source projects perform cross-cutting elements of microservice architectures such as authentication, service discovery, logging and monitoring, load balancing, scaling, and several more.

It is easier for application developers to offer alternative interfaces to their applications. If everything is an API, interactions within application components become standardized. All components have to make use of their application and data is to be able to verify and communicate over those standard APIs. Allows both those inside and, when relevant, outside your organization to simply generate new methods to use your application's data and services.

1.2.1 CLIENT-TO-MICROSERVICE COMMUNICATION

If it happens to client-to-microservice communication, an application's client calls all of the microservices straight utilizing the public endpoint (https://serviceName.api.dineshonjava.name) of that microservice. In the case of a clustered microservice, a URL would map to the load balance of a microservice.

The following diagram shows client-to-microservice communication:



As you can observe in the above diagram, the mobile and browser clients make requests to each of the services to recover order details. This method has several difficulties and limitations.

Client-to microservices communication is one of the challenges of the microservices architecture. In a microservices strategy, an application is classified into shorter sets of loosely coupled services. The limits of microservices are determined based on a decoupled applications domain model, data, and context. The real challenge is how to recover data for a business situation that requires requesting multiple microservices. An API gateway is one of the recommended resolutions for managing this challengeAPI Gateway In microservices architecture, each microservice presents a set of fine-grained endpoints. Typically, because of the quality of client applications, data required by a client includes the collection of data from different microservices. As a shade in a client application may want data from various microservices, a client application can relate to multiple microservices independently, which may point to certain issues.

- If a client app is a server-side web application like ASP.NET MVC, the communication to multiple microservices can be effective. With mobile and SPA clients, chatty communication to multiple microservices can make expenses due to network connectivity and the performance of the mobile devices.

- Due to the type of client applications, the data requirements of clients can be very strange; for example, a desktop-based version of an employee information page may give more than the mobile-based version.
- Every microservice is accountable for cross-cutting matters like authentication, authorization, logging, and throttling, which are significant expenses from a study and development viewpoint. While looking at these challenges, it can become a dream for a designer to achieve a client application requesting various microservices. One of the best feasible solutions to this difficulty is to execute an API gateway solution. At the upper level, an API gateway is a starting point service for a collection of microservices. It is very similar to the facade pattern in object-oriented design. It is essential to highlight that should a single custom API gateway service can be a danger if not implemented accurately. The gateway service develops and results based on the terms of the client apps. Finally, it can point to a situation similar to a monolithic application. Therefore, it is suggested to divide the API gateway within service categories; for example, one per client app, form factor type, and so forth. API gateways should never perform as a single aggregator for all internal microservices. It should be separated based on business margins. API gateways can produce various functions and characteristics, which can be arranged into the following design patterns.

1.2.2 GATEWAY AGGREGATION

As described in Figure 1.2, this design decreases chattiness within the client and the services. The gateway services become the entrance point. They take client requests and deliver those requests to the different backend microservices. The gateway including aggregates and joins the effects and transfers them back to the requesting client. This design works great if a client requires interfacing with various services for a particular business scenario and information collection is expected from multiple services. It also supports situations wherever the client is working on a high-latency network, such as mobile phones.

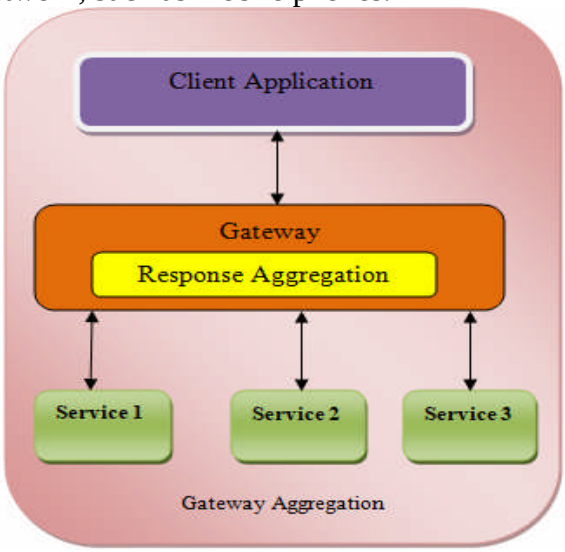


Figure 1-2. Gateway aggregation

The following points should be involved when executing this design.

- A gateway service should be placed near the back-end microservices to decrease network latency. A gateway service should not mix with services found over data centers.
- A gateway should not become a bottleneck. It should have the capacity to balance on its individual to maintain the application load
- Situations in which one of the services times out and partial data passed to the client application should be achieved correctly.
- Performance testing should create a gateway service that is not starting significant delays.
- Divided tracing should be approved with the help of similarity IDs to allow monitoring in case of negligence and for analysis.

1.2.3 GATEWAY ROUTING

As described in Figure 1.3, this design is similar to gateway aggregation, where, essentially, the gateway only routes the applications to multiple services applying a single endpoint. This design is helpful if you want to show many services on a single endpoint and direction to the appropriate service based on the client's request. In a situation where service is terminated or consolidated with other services, the customer can operate seamlessly except for an update as the intelligence of routing is controlled by the gateway, and modifications are needed just at the gateway level. In an industry situation, one use case for gateway routing is to present on-premise APIs to the outside society on the Internet. In a situation where you have to show an API to a companion, for a merchant that is not attached to a corporate network, an API gateway can show a public endpoint with the expected security and can internally route the traffic to on-premise APIs. Azure Application Gateway is a successful run load-balancing service that can perform a gateway routing pattern. To maintain this pattern, layer 7 routing is managed by the gateway to route the request to the relevant service. As gateways include a layer of abstraction, a deployment and service update can be efficiently managed. It also enables maintaining multiple services by adding a new route or by routing the traffic within an earlier or a fresher service endpoint without modifying the client.

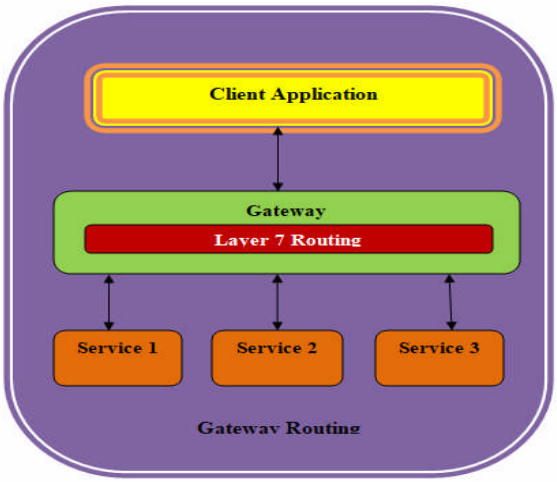


Figure1.3. Gateway routing

The following points should be executed while performing this pattern.

- The gateway service should be placed near the backend microservices to overcome network latency. The gateway service should nevermore associate with services placed over data centers.
- A gateway should nevermore grow a bottleneck. It should have the capacity to estimate on its individual to hold the application load.
- Performance testing should be performed to make assured that the gateway service is not including notable delays.
- A gateway route should help layer 7 routings. It can be based on the IP, port, header, or URL.

1.2.4 GATEWAY OFFLOADING

As described in Figure 1.4, the gateway offloading pattern helps offload the cross-cutting matters from unique microservices to the gateway service. It explains the implementation of each microservice as it combines cross-cutting interests into one-tier. With the guidance of offloading, specific characteristics can be performed by a functional team and at one basic tier. In conclusion, it can be used by each microservice.

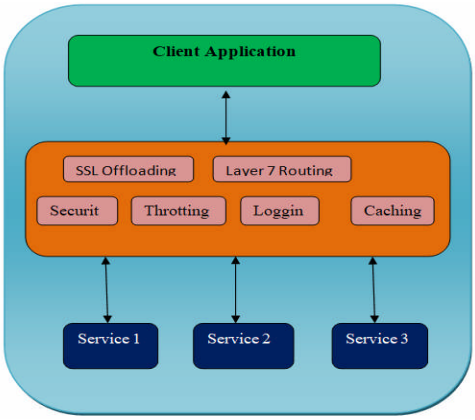


Figure1.4. Gateway offloading

Most of the common cross-cutting concerns that can be efficiently managed by the API gateway are

- Authentication and authorization
- Logging
- Throttling
- Service discovery
- SSL offloading
- Response caching
- Retry
- Load balancing

The following factors should be made while performing this pattern.

- API gateways can include a single point of failure.
- The scaling of an API gateway is necessary, unless it can become a bottleneck.

- If a company purposes to perform the gateway on its own, preferably with trained services like Azure API Gateway and APIM, it may want specific sources, which can significantly improve improvement efforts.

1.2.5 THE API GATEWAY PATTERN ON AZURE

There are several possible API gateway implementations, such as Kong and Mulesoft, and each gives a distinct subset of characteristics. We will describe the choices available on Azure.

- Azure Application Gateway. Application Gateway is a distributed load-balancing service that gives layer 7 routings and SSL end. It also gives a web application firewall (WAF).

- Azure API Management. API Management allows publishing APIs to external and internal customers. It provides specialties such as rate-limiting, IP whitelisting, and authentication using Azure Active Directory or other identity providers. Since API Management doesn't produce any load balancing, it should be used in combination with a load balancer such as Application Gateway or a reversed proxy.

1.3 INTERSERVICE COMMUNICATION

Inter service Communication

A component can request another component as they are operating in the same process, With monolithic applications, Also, the language level constructs (like new class name ()) can be utilized to request methods on another component. One of the challenges with microservices architecture is managing inter service communication as the in-process requests exchange to remote procedure requests. If a microservice is requesting another microservice, it breaks the fundamental principle of microservices. A basic principle of the microservices architecture is that every microservice is independent and available to the client, yet if the other services are sick or weak. There are various resolutions to this problem. One solution is to accurately fix the boundary of each microservice. This provides the microservice to be separated, autonomous, and independent of other microservices. Communication within the internal microservices should be minimum. If communication is needed, asynchronous communication should take precedence over synchronous communication because it decreases coupling within services. It also raises responsiveness and various subscribers can subscribe to the equivalent event. In asynchronous messaging, a microservice interacts with different microservice by giving messages asynchronously. If a return response is required, it appears as a separate message and the client believes that the response will not be collected quickly, or there may not be a reply at all. Asynchronous messaging and event-driven communication are crucial when creating variations over various microservices, and they are needed to obtain final agreement, as depicted in Figure 1.5.

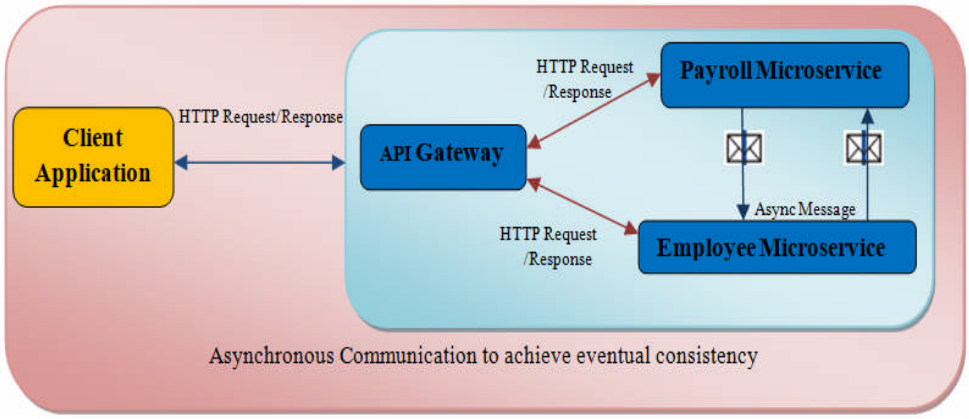


Figure1.5. Asynchronous communication to achieve eventual consistency

Asynchronous messages are normally based on asynchronous protocols such as AMQP. Message brokers are generally preferred for these sorts of communications (e.g., Rabbit MQ, N Service Bus, Mass Transit) or a scalable service bus in the cloud, like Azure Service Bus. If there is a requirement to query real-time data (e.g., to update the UI), usually, request/response communication with HTTP and REST is applied to help these varieties of situations. In this set of patterns, the client believes that the response will appear in a short time. If synchronous communication is needed among services, you can get a profit from binary format communication mechanisms (e.g., Service Fabric remoting or WCF using TCP and a binary format). You can also take advantage of caching using the cache-aside pattern. You should be careful of choosing this pattern because having HTTP dependencies within microservices gives them non-autonomous and appearance is affected as soon as one of the services in the series does not perform well. This architecture can be simply created and improved by applying technologies such as ASP.NET Core Web API, as depicted in Figure 1.6

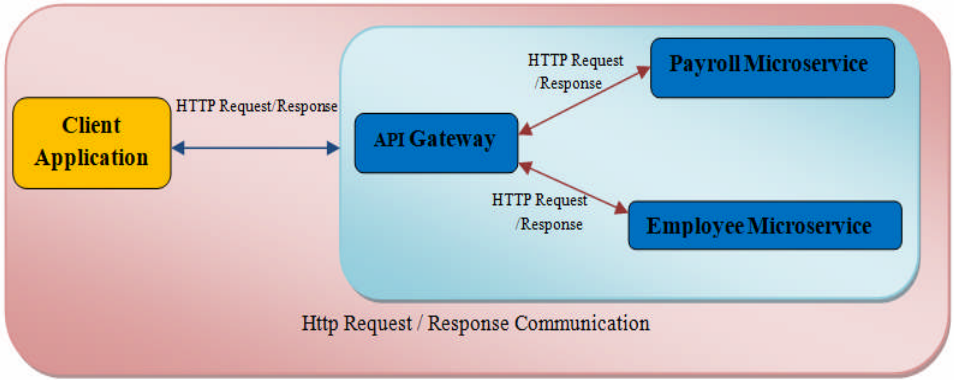


Figure1.6. HTTP request / response communication

1.4 DATA CONSIDERATION

Data Considerations

A fundamental principle of microservices is that all services should control their data. All services should be liable for their data store, and

other services should not obtain it directly. This limit coupling between services and provides the services to estimate based on capacity requirements. This principle also provides the services to manage various database technologies if needed. Due to the divided way of managing data, some challenges happen, similar to the repetition of data over data stores. One of the main challenges in delivering the updates over services because it is not possible to produce a database transaction over multiple services. There are various solutions to this difficulty. One solution is to cover ultimate consistency where possible; we should sharply separate use cases where ACID transactions are needed and where eventual consistency is acceptable. In situations where heavy consistency is required, one service should express the source of fact for a given entity. In scenarios where transactions are needed, patterns such as remunerating transactions can be utilized to store data consistent over several services. Ultimately, a good solution for this problem is to use final consistency within microservices through event-driven communication. In this architecture style, a service writes an event when there are changes to its public models or entities. Interesting services can subscribe to these events.

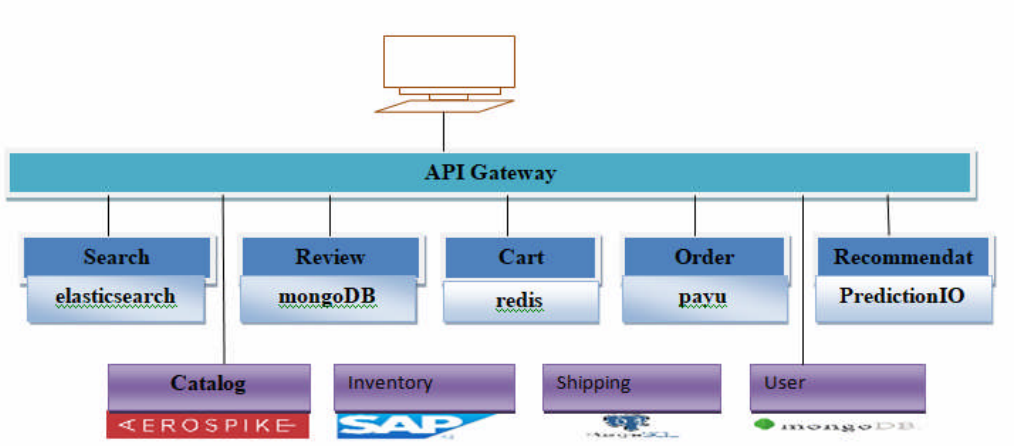
Common Database Techniques and Patterns Indexed shows the HRMS system client application that requires the appearance of employee's data along with payroll details, and there are two microservices involved (i.e., employee and payroll services). As per the primary principle, each microservice has its data, and the application reads and writes the data only via well-defined interfaces. As employee data will be required on almost all the screens of the client application, a denormalized read-only table can be created to be used only for queries. Since the view is generated in the improvement and includes denormalized data, it helps effective querying. An essential point to note is that the data in the indexed view is fully disposable because it can be completely restored from the source databases. Another classic use case that can be efficiently handled by this approach is replicating the master data that is needed by almost all the microservices. Having an HTTP call over the services or the database joins can be an ineffective approach from the performance and dependence aspect; therefore, indexed opinions efficiently resolve this problem. Here are some perfect reasons for applying this technique

- Indexed views significantly develop query review for recording and display needs.
- In cases where the reference data is ready in a normalized form and needs complex queries, producing an indexed show eliminates complexity while reading the data.
- It provides access to data based on privacy demands.
- It efficiently helps separate situations, in which the source database is not always available.

1.5 SECURITIES

1.5.1 Top 5 Microservices Security Challenges

The diagram here shows how microservices architecture includes various elements, each with its vulnerabilities and protection risks. Let's evaluate the top 5 microservices challenges getting it hard to ensure modern applications.



1. Infrastructure design and multi-cloud deployments

Microservices are divided over many data centers, cloud providers, and host machines. Raising infrastructure over many cloud environments improves the chance of missing control and clarity of the application components.

2. Segmentation and isolation

Decoupled application components play their responsibility in co-dependence with several other services. All these components build and support communication channels across various infrastructure layers, so usually, cross-service communication is bounded when examining for security vulnerabilities, the outcome of this is significant publicity in the interfaces between these services.

3. Identity management and access control

Microservices show different approach points to both internal and external actors. Access controls require to be adapted for all objects, whether legitimate or illegitimate. It is necessary to have an organizational interface that can support you control users, applications, groups, devices, and APIs from one primary location, providing real-time clarity into what is occurring in the environment.

4. Data management

Data created in microservices architecture progress, changes, and is continuously associated with data is also saved in separate places and for different purposes. Buyers of data assets want insight into the life cycle and the dynamics of data to avoid gaps.

Can you be assured that **your data is secure?**

Data losses can happen carelessly of the communication channel's presentation. Malicious actors can chain vulnerabilities to crack through to individual assets.

5. The rapid rate of application changes

Application improvement in modern SDLC makes the code base and data stores increase across time. Improvement methodologies promote iterative and incremental growth, setting microservices below the constant workload. How can you identify at any time that new code getting through the growing pipeline will not disclose your application to the new sets of vulnerabilities and serious attack vectors? Protection examination must store up with the movement of the SDLC, to improve Dev Sec Ops.

Decomposing applications into microservices improves the application's attack surface because of recently joined entry details and relationships among instances that are now spread out across many environments, because of that microservices protection needs non-trivial and ready-made solutions.

1.5.2 DEPLOY SECURITY AT CONTAINER LEVEL

Microservices are based on container technology. The obvious vulnerability of containers is that they are based on images, which may contain vulnerabilities. Conduct regular scanning to use images that contain security vulnerabilities or other security issues.

A container has both internal and external warning surfaces. A first step to preserving containers at runtime is using the principle of shortest privilege (POLP). This can include some or all of the following strategies:

- Limit support to the minimum wanted by each user or service role
- Never use privileged accounts to run services
- Control the use and available resources—for example, restrict a container's way to the owner operating system
- Never store secrets on the container, because anyone with access to the container can see them
- Define isolation with proper rules for resources with various levels of consciousness

1.5.3 CREATE AN API GATEWAY

Microservices often cross several different networks, utilizing a change of technologies, interfaces, and protocols. An essential way to guard microservices is to produce one entry point, which all clients and systems obtain, and which can easily be secured.

This entry point is called an API gateway. You have to design your system in such a way that all clients regularly join the API gateway, also you can use it to perform authentication and filter requests to sensitive devices. The API gateway authenticates users and service roles and decides which microservices they are allowed to access, also provide additional security capabilities like SSL termination, protocol conversion, monitoring, routing, and caching of requests.

1.5.4 ISOLATION

In a microservices architecture, each service is a separate, isolated part of the application. You should be able to implement, maintain, modify, extend and update microservices without influencing other nearby microservices. Isolation should also be performed at other layers of the support, such as the database. One microservice should not have entrance to data relating to other microservices. By obtaining complete isolation at all layers, criminals who compromise one microservice cannot perform the lateral movement to attack other parts of the system.

1.5.5 DON'T SHOW SENSITIVE DATA AS PLAIN TEXT

The plaintext can be quickly read and copied by users and machines. A primary step to guarding personally identifiable information (PII) is to avoid exposing it in cleartext. All passwords

And usernames must be removed or hidden when saving accounts or records. Adding TLS/HTTPS will not resolve the problem if data is saved insecurely at rest. Also, you can try to encrypt logs, but this will support attackers who can directly access system memory. Therefore the only reliable way is to avoid persevering sensitive information in the first place.

1.5.6 AUTOMATED MICROSERVICES SECURITY WITH NEXDAST

Nex DAST integrates automated AI-powered Dynamic Application Security Testing into the SDLC to examine applications made at the top of the difficult microservices architecture every time the application code is packaged to a functioning application and delivered to the testing phase.

Nex DAST presents with real-time reports with zero false positives in no time, where we can see reported vulnerabilities with a reproducible proof-of-concept and ordered by the sharpness of the impression on an application, that proves the overall strength and ability of an application to control product runtime without being displayed to vulnerabilities, careless of the field and the complexity of the underlying microservices architecture mesh.

1.6 MONITORING

The requirement for microservices can be defined as speed. The need to give more functionality and security faster has changed the way developers build software. This change has created ripple impacts in software management, including monitoring systems. Monitoring is a crucial part of the control systems of microservices, as the more complicated software gets, the more difficult it to understand its performance and troubleshoot problems. The five principles of monitoring microservices, as follows:

1. Monitor containers and what's inside them.
2. Alert on service performance, not container performance.

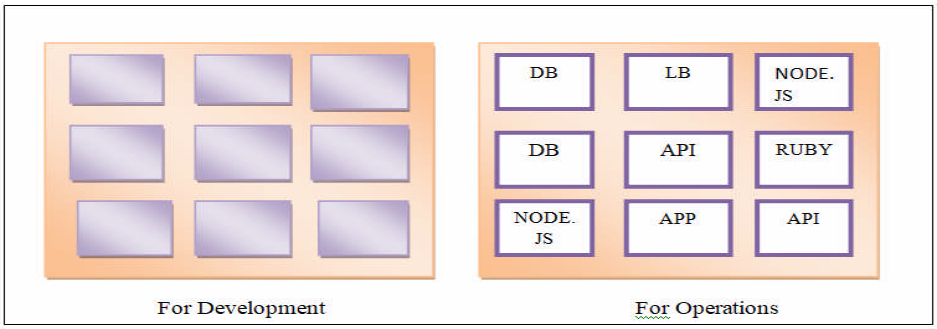
3. Monitor services that are elastic and multi-location.
4. Monitor APIs.
5. Map your monitoring to your organizational structure.

Leveraging these principles will enable you to build more efficient monitoring as you produce your way towards microservices. Those principles will support you to address both the technological changes connected with microservices and the organizational changes related to them.

1.6.1 THE PRINCIPLES OF MICROSERVICE MONITORING

1. Monitor Containers and what's running Inside Them

The speed, portability, and isolation of containers created it simple for developers to include a microservice model. Containers are black boxes to most systems that exist around them. That is especially helpful for development, allowing a high level of portability from development through production, from developer system to cloud. But for operating, monitoring, and troubleshooting services, black boxes make regular activities harder, what's running in the container? How is the application/code performing? Is it ejecting out major custom metrics?



The ideal method for instrumentation in non-containerized conditions — an assistant that exists in the user space of a host or VM — does not operate well for containers, because containers profit from being small, separated processes with as few dependencies as possible. And, running thousands of monitoring agents for even a modestly-sized deployment is a valuable method of resources and an orchestration vision. Two possible resolutions stand for containers:

1) ask your developers to instrument their code directly, or
2) leverage a universal kernel-level instrumentation way to see all application and container activity on the owners.

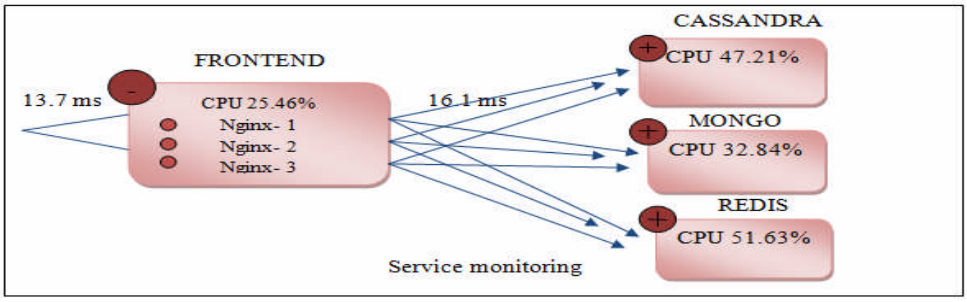
2. Leverage Orchestration Systems to Alert on Service Performance

Creating a feeling of operational data in a containerized environment is a new challenge. The metrics of a single container have a much smaller minimal value than the aggregate data from all the containers that make up a service. Especially applies to application-level information, like which queries have the most delayed response times or which URLs are viewing the most mistakes, but also refers to

infrastructure-level monitoring, such as which services' containers are using the most resources beyond their allocated CPU shares.

Frequently, software deployment needs an orchestration system to "translate" a logical application design into physical containers. Popular orchestration systems introduce Kubernetes, Mesosphere DC/OS, and Docker Swarm. Companies adopt an orchestration system to (1) define their microservices and (2) get the current state of each service in deployment. The original containers are temporary — they involve only for the short time they exist — while services concern for the life of their usefulness. DevOps organizations should redefine warnings to concentrate on characteristics that get as close to monitoring the experience of the service as reasonable. These signals are the primary line of protection in evaluating if something is changing the application. But learning to these alarms is challenging, if not possible except your monitoring system is container-native.

Container-native resolutions support orchestration metadata to dynamically aggregate container and application data and determine monitoring metrics on a per-service base. Depending on your orchestration device, you force have several layers of a hierarchy that you like to drill in it. For instance, Kubernetes, typically have a Namespace, Replica Sets, Pods, and some containers. Aggregating at those different layers is necessary for logical troubleshooting, although the natural deployment of the containers that builds up the service.



3. Be Prepared for Services that are elastic and Multi-Location

Elastic services are not a new theory, but the velocity of innovation is much quicker in container-native environments than in virtualized environments. Quickly growing environments can wreak destruction on crisp monitoring systems. Regularly monitoring legacy systems needed manual tuning of metrics and checks based on singular deployments of software. This tuning can be as precise as representing the unique metrics to be achieved or configuring selection depend on what application is working in an appropriate container. While that may be satisfactory on a small scale, it would be unacceptable in anything. Microservice-focused monitoring must be capable to easily develop and shorten in step with elastic services, without human interference.

For instance, if the DevOps organization must manually determine what service a container is involved in for monitoring persistence, they

drop the ball as Kubernetes or Mesos rotates up new containers frequently during the day. Likewise, if Ops were required to install a system stats endpoint when new code is created and launched into production, difficulties may appear as developers tend to base images from a Docker registry.

In production, establish monitoring to a complicated deployment that crosses multiple data centre's or multiple clouds. Leveraging, for instance, AWS CloudWatch will only take you so far if your services cross your private data center as well as AWS. That points back to performing a monitoring system that can cross those various locations and perform in dynamic, container-native environments.

4. Monitor APIs

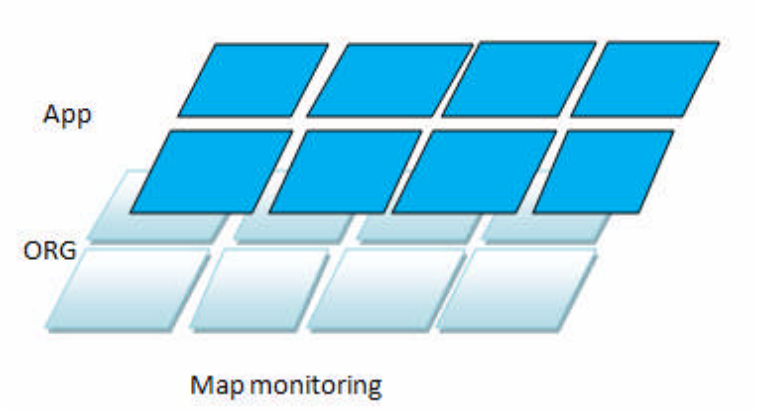
In microservice environments, APIs are the lingua franca. They are the only components of a service that are presented to other organizations. Acknowledgment and compatibility of the API may be the "internal SLA",

As a consequence, API monitoring is necessary. API monitoring can get many forms but obviously, must go away with binary up/down controls. For example, it's worthwhile to know the most commonly utilized endpoints as a use of time. This enables organizations to view if anything notable has changed in the method of services, it is due to a design modification or a user modification.

Lastly, the capacity to track service calls through the system describes another crucial ability. While handled by developers, this type of profiling will help you understand the overall user experience while developing information down into infrastructure and application-based views of your environment.

5. Map Monitoring to Your Organizational Structure

Conway's law, suggests that the purpose of systems is determined by the organizational structure of the organizations making them. The appeal of creating faster, more flexible software has forced organizations to study restructuring their development organization and the laws that govern it.



If an organization needs to profit from this software architecture program, its organizations must reflect microservices themselves. It intends smaller organizations, loosely coupled; that can take their direction as long as it yet satisfies the poverty of the whole. In each organization, there is more power than always above languages managed, how viruses are checked, or even operational responsibilities.

DevOps organizations can allow a monitoring program that does specifically this: enables each microservice organization to divide their signals, metrics, and dashboards, while yet providing services a look into the global system.

1.7 MICROSERVICES HOSTING PLATFORM OPTIONS

An essential section of the study is whether to host microservices on virtual machines or containers. Both choices can be executed on-premise and on cloud platforms like Microsoft Azure. On Azure, the administration of an underlying infrastructure becomes very simple because there are specific services ready, such as Azure Serve Fabric and Azure Kubernetes Service (AKS). Utilizing containers to execute microservices is the most selected choice, and it is necessary to know the many causes behind it.

A virtual machine is an operating system installation on the virtualization layer of the physical manager, as described in Figure 2-6. Managing Virtual machines helps to optimize the hardware utilization by allowing the physical manager to give a private environment for each application. The warning is that for all virtual machines, an entire OS must be installed individually. Hence, every virtual machine requires to boot up and load OS files into its memory. This device consumes lots of computing sources on the host operating system. Figure1.11. Shows Virtual machine hosting

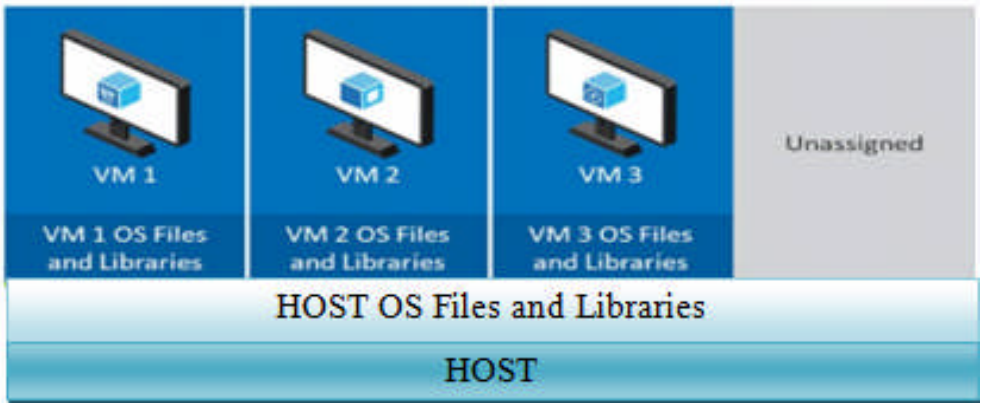


Figure1.11. Virtual machine host in

Therefore, every virtual machine needs to boot up and load OS files into its memory. This mechanism dissipates lots of compute resources on the host operating system.

1.7.1 USING A CONTAINER

Containers are similar to virtual machines. They give an idea to protect an application into its separate box utilizing namespace separation. In this procedure, the host OS generates a namespace for all the devices (e.g., disk, memory, running process, etc.) to create an environment that looks as if applied to the container. Containers vary from virtual machines in a few methods.

- Virtual machines hold an entire OS installation on the virtualization layer of the physical host. It needs time to rise because it must boot the complete OS and map OS files in the memory. Containers give the same kernel, so there is no requirement to boot the OS and map files to the memory (see Figure 1.12). Hence, a container step is small as opposed to virtual machines, and they boot up in a much less time

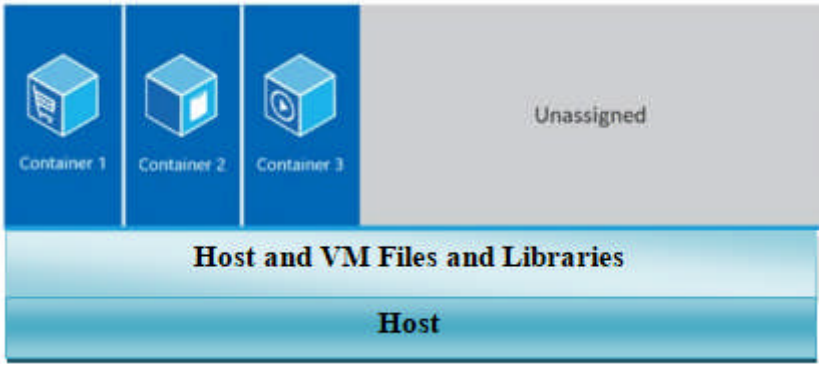


Figure1.12. Containers sharing OS files and libraries

Beginning with Windows 2016, there was a choice to host containers in Hyper-V mode, which divides the kernel of a container from the host OS. It can be applied for extremely sensitive applications or multitenant environments. Start-up capability decreases when opposed to containers that utilize namespace isolation.

- As containers create the environment and resource using regular, it becomes available for developers to operate the same application on various systems without modification inability. By virtual machines, still, applications can work separately in different environments. Microservices divide a complete resolution into multiple services; activity, scalability, and optimum resource utilization are the most significant portions. After containers work much better than VMs for such situations, they are an enterprise's prime choice for a deployment stage. Let's glance at the fundamental components of a container ecosystem.
- Container image
- Container registry
- Container
- Container Image

A container image is similar to a software installer file that includes both the OS layer and the application layer, with all the

dependencies to run the application (e.g., Windows Nano Server, SQL Server). A container image can be utilized many times to install an application.

1.7.2 CONTAINER REGISTRY

The container registry is the depository for the container models that can be made available to the whole organization. Any approved user in an organization can push or pull images from this container. It can be performed as either public or private, depending on the demands. Microsoft Container Registry is a public registry that hosts images for public download. Azure Container Registry is utilized for keeping a private registry. The command to download an image from a public repository is docker pull mcr.microsoft.com/mssql/server:2017-latest Let's break down this command

- mcr.microsoft.com/mssql/server:2017-latest is the container image.
- mcr.microsoft.com/mssql/server is the container registry.
- 2017-latest is the tag.
- Docker pull – docker is the command line to pull an image from the registry.

1.7.3 CONTAINER

The container is an example of a container image. Multiple container cases can be formed from a single container image. If a SQL Server container is formed from the image of the local container, it will produce an example of SQL Server on an Ubuntu server. The practice is like a virtual machine, where you can take in the OS layer, run commands, and work with SQL Server from both inside and outside of the containers. The selection of Orchestrators plays an important function in maintaining a large number of containers or virtual machines. High availability, scalability, and application activity are the most significant parts that orchestrators are supposed to cover. The following are the most important functionalities that an orchestrator should cover.

- Clustering of resources. This characteristic creates groups of VMs or physical machines that look like a single resource. All the sources are given by a single group. This serves to optimize resource utilization, and even management becomes simple.
- Orchestration. These characteristic benefits perform all the elements work collectively as a unit. Running containers, their scalability, load balancing when a heavy load, and high availability while losers are secured by this functionality.
- Management. Managing networking, storage, and services come below the management functionalities of the orchestration agents. Here are a few of the orchestration solutions prepared on the market.

- Azure Service Fabric
- Azure Kubernetes Service

- Docker Swarm and Compose
- Mesos DC/OS

Let’s an overview of these solutions

Azure Service Fabric

Azure Service Fabric is an orchestration agent that can be used both on-premises and on Microsoft Azure. This Microsoft resolution maintains various services on Azure. Applications are used in the form of services on Azure Service Fabric. Each service (stateless or stateful) has three elements.

• Code
• Configuration
• Data

A Service Fabric cluster is established on a group of physical or virtual machines named nodes. There are several services (e.g., failover manager services, repair manager services, naming services, etc.) to maintain high availability, health, and service description for Azure Service Fabric. Apart from containers, services can be run as fellow executables and dependable services by utilizing the local Service Fabric SDK.

Azure Kubernetes Service

Kubernetes is an orchestration agent that can be used both on-premise and on Microsoft Azure. On Azure, it is a controlled service named Azure Kubernetes Service. On AKS, seeds run a single or a group of containers. Services are the labels that point to multiple pods. Kubernetes has a cluster master and cluster nodes to maintain a container ecosystem.

Docker Swarm and Compose

Docker Swarm is the clustering agent for Docker; it can be used both on-premise and on Microsoft Azure. Each node in the cluster operates as a swarm agent, and one of the nodes runs a swarm manager. A swarm manager is accountable for orchestrating and maintaining the containers on the possible container’s host. Filters can be configured on Docker Swarm to manage the hosting of containers. Docker Compose is a command-based service to configure an application’s services. With a single command, a complete application can be up and operating on the swarm cluster.

Mesos DC/OS

The Apache Mesos orchestration resolution is created to manage a large number of hosts to help different workloads. It can be run both on-premise and on Microsoft Azure. This setup has a Mesos master to orchestrate the tasks, and agent nodes to perform the tasks. Frameworks coordinate with the master and plan tasks on agent nodes.

LET US SUM UP

There is one, pure trigger event that accelerated the movement to microservices: speed. Organizations wanted to give more abilities to their clients in a shorter time. Once this happened, technology moved in, the architecture movement to micro-services, and the underlying shift to containers perform speed to follow. Anything that receives in the process of this development series is working to become run over on the tracks. As a consequence, the primary principles of monitoring require adapting to the underlying technology and organizational settings that bring microservices. Services organizations that understand this group can adapt to microservices earlier and easier.

LIST OF REFERENCES

- https://www.jigsawacademy.com/blogs/cloud-computing/what-is-microservices/
- https://opensource.com/resources/what-are-microservices
- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri
- https://www.neuralegion.com/blog/microservices-security/
- https://thenewstack.io/five-principles-monitoring-microservices/

BIBLIOGRAPHY

- https://www.jigsawacademy.com/blogs/cloud-computing/what-is-microservices/
- https://opensource.com/resources/what-are-microservices
- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri
- https://www.neuralegion.com/blog/microservices-security/
- https://thenewstack.io/five-principles-monitoring-microservices/

UNIT AND EXERCISES

1. Why consider API Gateways instead of direct client-to-microservice communication.
2. What is the API Gateway pattern?
3. Explain the features of API Gateways.
4. Write down the Drawbacks of the API Gateway pattern.
5. Which interface is used for inter-service communication?
6. What are the features of Microservices?

7. What are the considerations of implementing Microservices?
8. How do you maintain data consistency across Microservices?
9. How would you implement security in Microservices?
10. Why are Container used in Microservices?

MCQs

I. What is a microservice?

a) Design used primarily in functional programming and object oriented programming
b) A small program that represents discrete logic that executes within a well-defined boundary on dedicated hardware
c) A style of design for enterprise systems based on a loosely coupled component architecture
d) A very small piece of code that never gets any bigger than 10 lines

II. When would developers use microservices?

a) When they want to write cell phone applications that run quickly
b) When they work with ephemeral nano technology
c) When they need to create large, enterprise-level applications that are subject to changes on a frequent basis
d) When they create applications specifically for scientific test equipment

III. Which of these concepts are essential to microservices security?

a) Manual human testing
b) Automation
c) Department siloing
d) Removing security policies from the development pipeline

IV. Which of these elements should be a part of your microservices strategy?

a) Service mesh
b) Thorough access management
c) Threat detection in production
d) All of the above

❖❖❖❖

2

AZURE SERVICE FABRIC

Unit Structure :

2.0 OBJECTIVES

- Manage Azure Service Fabric to resolve infrastructure orchestration challenges
- Study of software concepts related to Service Fabric, including collections, the actor model, and stateful vs stateless services
- Use an application to a Service Fabric cluster
- Azure Service Fabric plans to give developers a very strong platform that addresses several complexities that are associated with developing cloud-based distributed applications.
- “By utilizing Service Fabric developers and administrators can bypass resolving difficult infrastructure queries and concentrate preferably on implementing mission-critical, requiring workloads understanding that they are scalable, reliable, and manageable.” — Mark Fussel, principal program manager at Microsoft
- Service Fabric allows microservices to fast and efficiently. Service fabric-safe services are strong concepts, supporting both lifting and shifting applications, allows scaling, disaster recovery, and resilience.

2.1 INTRODUCTION

Azure Service Fabric is a distributed systems platform that enables you to operate, maintain, and scale applications in a cluster of nodes, utilizing any OS and any cloud. The Service Fabric SDK enables you to implement service communication, scale, and service discovery models efficiently. The SDK is ready for .NET and Java developers. You can generate an application in any programming language and can use Service Fabric to handle customer executables and containers. Service Fabric can be expanded on the platform of your choice (i.e., Windows or Linux) and can be extended on-premise, on Azure or AWS, or any other cloud platform. In-Service Fabric, an application is a set of various services, and every service has a specified purpose to perform. A service is described by three components: code (binaries and executables), configuration, and data (static data to be consumed by service). Every component is versioned and can be upgraded separately. This is the important advantage of Service Fabric—in a deployment failure, you can simply go back to an earlier version of the service. Also, note that a deployment and version upgrade is required even a simple modification to an application's configuration is done, in our knowledge, this is previously difficult for deployment organizations because most companies have powerful deployment methods, but it supports the matter of deployment crashes because you can go back to any of the earlier versions by applying a single command. Deployment methods are streamlined by using services like Azure DevOps. Service Fabric also helps autoscaling. The autoscale characteristic provides Service Fabric to dynamically scale your services based on pointers like load, sources rule.

2.2 AN OVERVIEW

It is simple to package, deploy, and manage scalable and reliable microservices and containers by using a distributed systems platform like Azure Service Fabric. Service Fabric addresses the important challenges in developing and managing cloud-native applications.

A fundamental differentiator of Service Fabric is its sharp focus on creating stateful services. You can utilize the Service Fabric programming model or run containerized stateful services written in any language or code. You can generate Service Fabric clusters anywhere, including Windows Server and Linux on-premises and other public clouds, along with Azure.

Service Fabric controls many Microsoft services, including Azure SQL Database, Azure Cosmos DB, Cortana, Microsoft Power BI, Microsoft Intune, Azure Event Hubs, Azure IoT Hub, Dynamics 365, Skype for Business, and many-core Azure services.

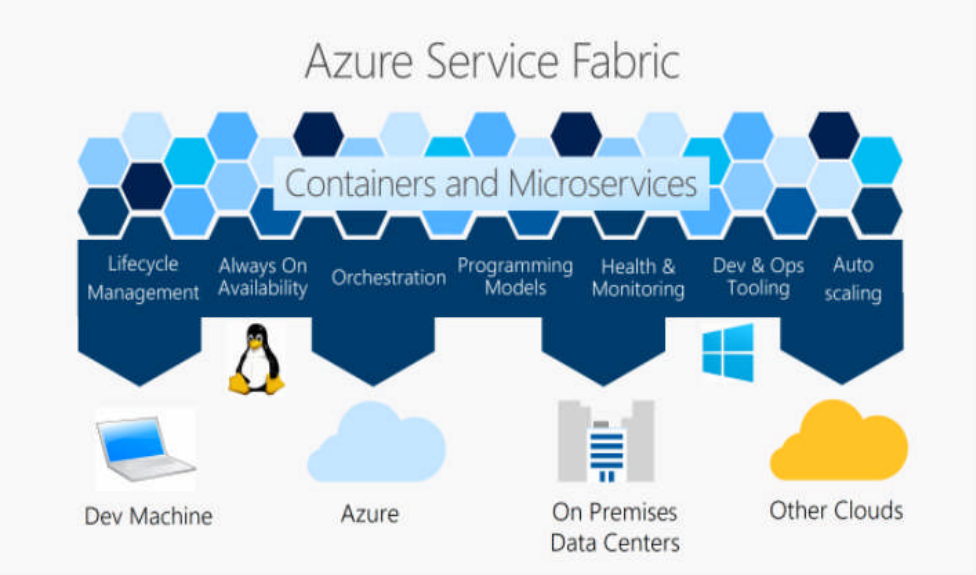


Figure: 2-1. Azure Service Fabric [4]

CONTAINER ORCHESTRATION

Service Fabric is Microsoft's container orchestrator for deploying and managing microservices over a cluster of machines, profiting from the models studied working Microsoft services at a massive scale. Service Fabric can deploy applications in seconds, at high mass with hundreds or thousands of applications or containers per machine. Using Service Fabric, you can join both services in rules and services in containers in the corresponding application.

STATELESS AND STATEFUL MICROSERVICES

Service Fabric gives a modern, lightweight runtime that helps stateless and stateful microservices. An important differentiator of Service Fabric is its strong support for creating stateful services, either with Service Fabric built-in programming models or containerized stateful services.

APPLICATION LIFECYCLE MANAGEMENT

Service Fabric gives help for the full application lifecycle and CI/CD of cloud applications including containers: development for deployment, daily monitoring, management, and maintenance, to final decommissioning. Service Fabric is combined with CI/CD agents such as Azure Pipelines, Jenkins, and Octopus Deploy and can be applied with any other current CICD tool.

ANY OS, ANY CLOUD

We can build clusters for Service Fabric in various environments, including Azure or on-premises, on Windows Server or Linux. We can also build clusters on other public clouds. The development environment in the Service Fabric SDK is equal to the product environment, with no emulators included. For Windows development, the Service Fabric .NET SDK is combined with Visual Studio and PowerShell. For Linux development, the Service Fabric Java SDK is mixed with Eclipse, and

Yeoman is used to making templates for Java, .NET Core, and container applications.

2.2.1 CORE CONCEPTS

In this segment, we explain core Services Fabric concepts, such as the application model, scaling techniques, supported programming models, and types of Service Fabric clusters.

2.2.1.1 Service Fabric Application Model

In-Service Fabric, an application is a set of services where each service and application is determined utilizing a visible file. All service in an application is described by a service package, and the package has three components (code, configuration, and data), as represented in Figure 2-2. The code element includes the real executables, binaries of the service, or signals to the container images in container closets so as ACR and Docker Hub. The configuration element includes the configuration approaches needed by the service; it's very related to the web. config in ASP.NET applications, and if required, you can hold various configuration files. The information component includes static data to be handled by the service. Service Fabric is not very accurate concerning the data form; it can be JSON, XML files, and so forth. Every component is versioned and can be upgraded separately. And, the service package is regularly used as a collection, which means if you need to create two containers worked together on the related node, you can hold two code packages (pointing to the respective containers) in the same service package

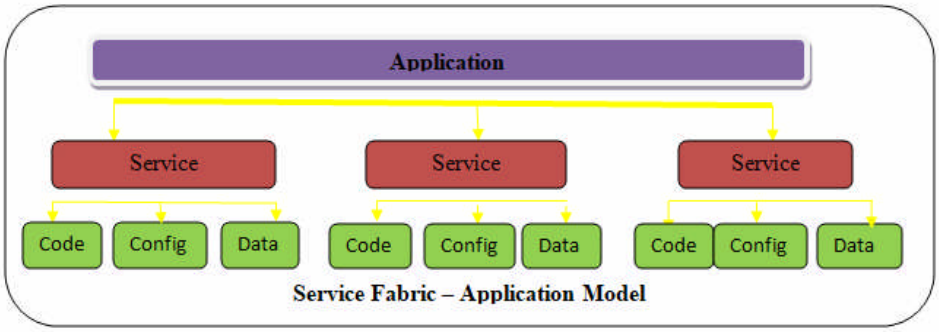


Figure: 2-2. Service Fabric application model

While using an application on Service Fabric, an Application Type gets created to represent an application and a Service Type gets created to describe all services in a Service Fabric cluster. Similarly, on successful deployment, an example of Application Type and Service type gets produced. We can have many examples of an Application type to maintain various versions of the related application and can have various instances of service type to support the higher load and high availability.

2.2.1.2 Scale by Increasing or Decreasing Stateless Service Instances

During generating a service instance, you can define the instance count, which determines the quantity of Service Fabric cluster nodes where the service is hosted. The resulting command defines the count to two, which indicates that the service is hosted only on two nodes, yet the

number of nodes in the Service Fabric is bigger than two (see Figure 2-3). sfctl service create –-name fabric:/a/s1 –-stateless –instance count

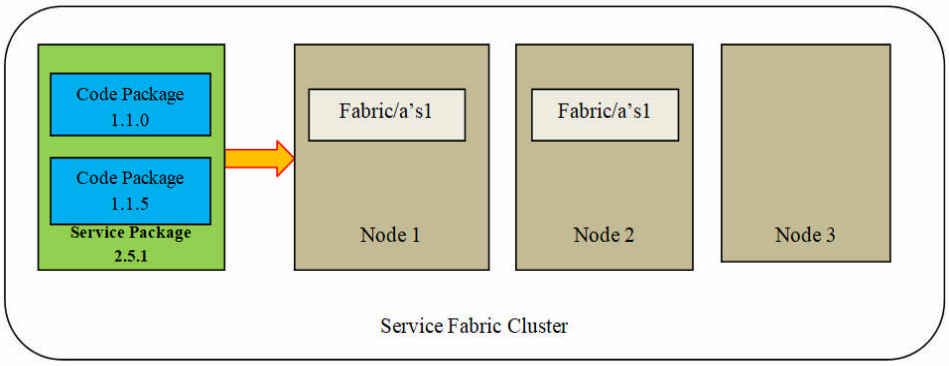


Figure: 2.3. Service Fabric Cluster: :two nodes utilized

Service Fabric also provides to renew the instance count. You can set the count to –1 to notify Service Fabric to continue the service on all possible nodes, as depicted in Figure 2-4. If new nodes are joined to the cluster, Service Fabric assured that service is received on the newly added nodes too. sfctl service update –-service-id a/s1 –-stateless –instance count -1

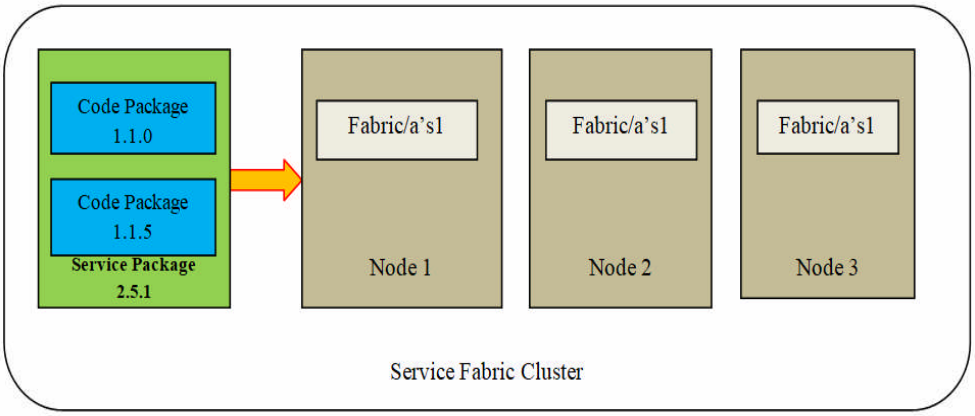


Figure: 2-4. Service Fabric cluster: all nodes utilized

2.2.1.3 Scale By Adding or Removing Named Services Instance

Order by Adding or Removing Named Services Instances In situations wherever the node ability is underutilized, we can direct Service Fabric to scale up by building another named service instance and use the corresponding code package on all the possible nodes, but with various uncommon names (see Figure 2-5). sfctl service update –-service-id a/s2 –-stateless –instance count –

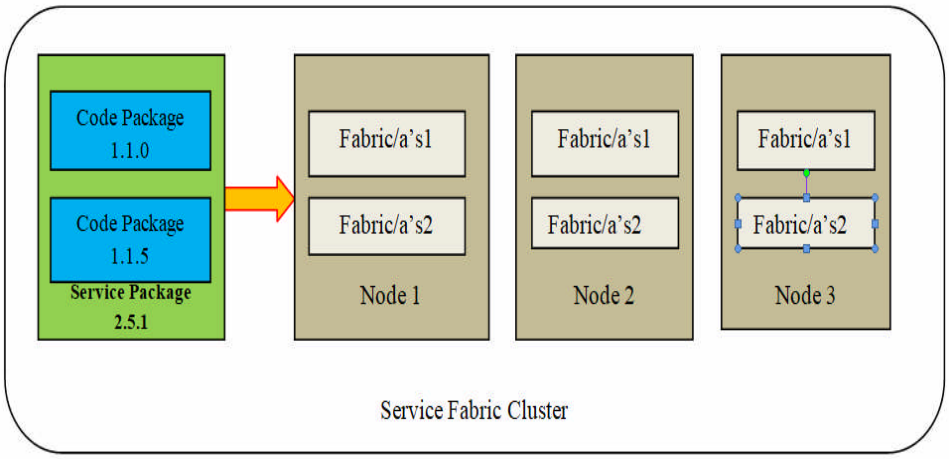


Figure: 2.5. Service Fabric Cluster: named service instances

2.3 SUPPORTED PROGRAMMING MODELS

Service Fabric supports the programming models shown in Figure 2-6.

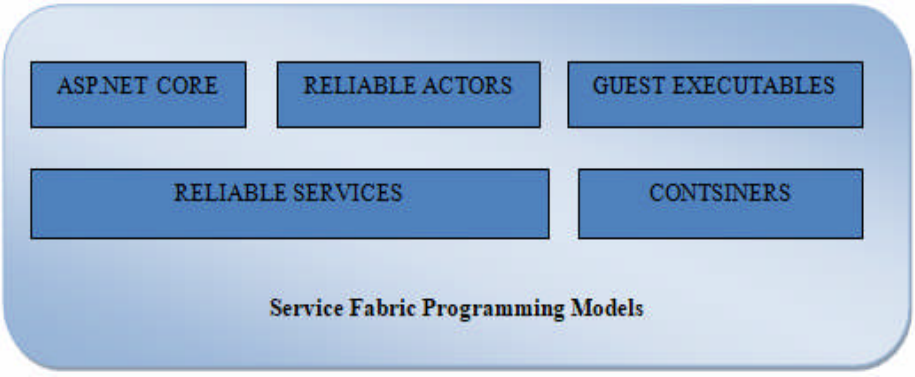


Figure: 2-6. Service Fabric cluster: supported programming models

Containers

One of the programming models offered by Service Fabric enables orchestration and use of applications utilizing both Windows and Linux containers. Service Fabric helps the deployment of Docker containers on Linux and Windows server containers (including Hyper-V isolation) on Windows Server 2016. Service Fabric can eliminate the container images from container repositories like Docker HUB and Azure Container Registry. Using an application as a container does not need any modifications to the application and has no Service Fabric SDK dependence.

Although you can use services using various programming models (guest executables, stateless or stateful services), there are several situations where containers are more proper.

Monolithic Applications

If a monolithic application is generated utilizing ASP.NET web forms and has a dependence on technologies like IIS, you can package those applications as container models and use them on Service Fabric for

efficient scaling and deployment management. In this method, you have no mandate on Service Fabric SDKs; you can use an application as it is. An application can also be produced in a programming language.

Application Isolation

If a higher level of separation of different applications working on the same host is needed, containers are a highly viable alternative because they give separation efficiently. Also, Windows Containers Hyper-V mode uses isolation to various levels because the core OS kernel is not given between containers. Service Fabric also gives resource governance abilities to reduce the devices that can be managed by a service on a host

Reliable Services

Good services enable you to write services utilizing the Service Fabric SDK framework. Service Fabric supports to control the life cycle of your services, it is a lightweight framework. It further supports the services to communicate with the Service Fabric runtime. With SDK, you can get profit from highlights such as notifications on code or configuration modifications and communicating with other services

2.4 SERVICE FABRIC CLUSTERS

Service Fabric Clusters

Service Fabric supports you to use microservices on a cluster, those are the set of virtual or physical machines that are connected by a network. Every machine within the cluster is named a node. A cluster can build up thousands of nodes, based on the source requirements of your application. Every node in a cluster has a Windows service named FabricHost.exe, which makes assured that the other two executables (Fabric. exe and FabricGateway.exe) are regularly operating on the cluster nodes. Service Fabric clusters can be generated utilizing virtual machines or physical machines operating on Windows or Linux. The Service Fabric cluster can run on-premise, on Azure, or any cloud (e.g., AWS). We want to have at least five nodes to run a Service Fabric cluster for production workloads.

Service Fabric has some system services to provide the platform capabilities that are as follows.

Naming Service

A naming service determines the service name to a location. For applications in a cluster that can transfer from one node to another, a naming service gives the real port and IP address of the machine where the service is running.

Image Store Service

When implementing a deployment, the application packages are uploaded to an image store, and then an application type is recorded for the uploaded application package

Failover Manager Service

As the name implies, the failover manager service is effective for the high availability and appropriateness of services. It orchestrates application and cluster upgrades.

Repair Manager Service

The repair manager service is an elective service to perform repair operations on silver and gold durability Azure Service Fabric clusters.

Cluster on Azure

By the Azure portal or by utilizing a source template a Service Fabric cluster on Azure can be generated. A Service Fabric cluster can be quickly generated by applying the Azure portal user interface. As the cluster and its components are similar to any other resource manager source, we can efficiently track access, cost, and billing. There are two important advantages of hosting a Service Fabric cluster on Azure.

- It appears with autoscaling functionality.
- It helps the installation of Service Fabric clusters on Linux machines.

Standalone Cluster or Any Cloud Provider

Deployment on-premise or on any cloud provider is much related. Service Fabric clusters can be generated utilizing the Windows Server 2012 R2 and Windows Server 2016 operating systems. Standalone clusters are helpful in situations where you can't hold your applications hosted on the cloud due to regulative or agreement constraints.

2.5 DEVELOP AND DEPLOY APPLICATION OF SERVICE FABRIC

Till now, we have reviewed Service Fabric and its programming models and studied that it can be installed on the cloud or on-premise. Now build some units to better explain that how we produce and use applications on Service Fabric. Here two samples will be cover.

- Situation 1. Express developing an ASP.NET Core stateless web app interacting with an ASP.NET Core stateful API.
- Situation 2. Express developing a Java Spring Boot application adopting Visual Studio Code and use it on Service Fabric as a visitor executable or as a container.

Develop an ASP.NET Core Stateless Web App

We will produce a simple ASP.NET MVC–based application to control operators. The ASP.NET MVC front end communicates with the ASP.NET API to execute CRUD operations. Inside, the Web API utilizes strong groups to save operator data.

Setting up the Development Environment

1. Install Visual Studio 2017.
2. Install the Microsoft Azure Service Fabric SDK.

3. Make assured that the Service Fabric local cluster is running. Secure this by browsing

HTTP:// localhost:19080/Explorer/index.html#/

or by right-clicking the Service Fabric icon in the system tray, as shown in Figure 2-7

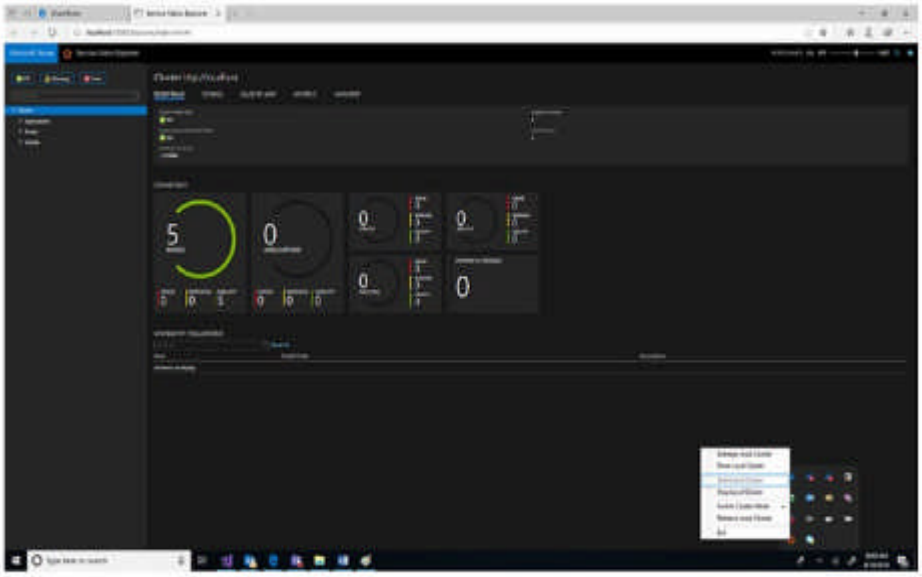


Figure: 2-7. Service Fabric status [3]

Create a ASP.NET Core Web API Using Reliable Collections

Following are the levels.

1. Launch Visual Studio 2017 as an administrator.
2. Create a project by selecting File ➤New ➤Project.
3. In the New Project dialog, choose Cloud ➤Service Fabric Application.
4. Name the Service Fabric application Employee (as depicted in Figure 2-8) and click OK.

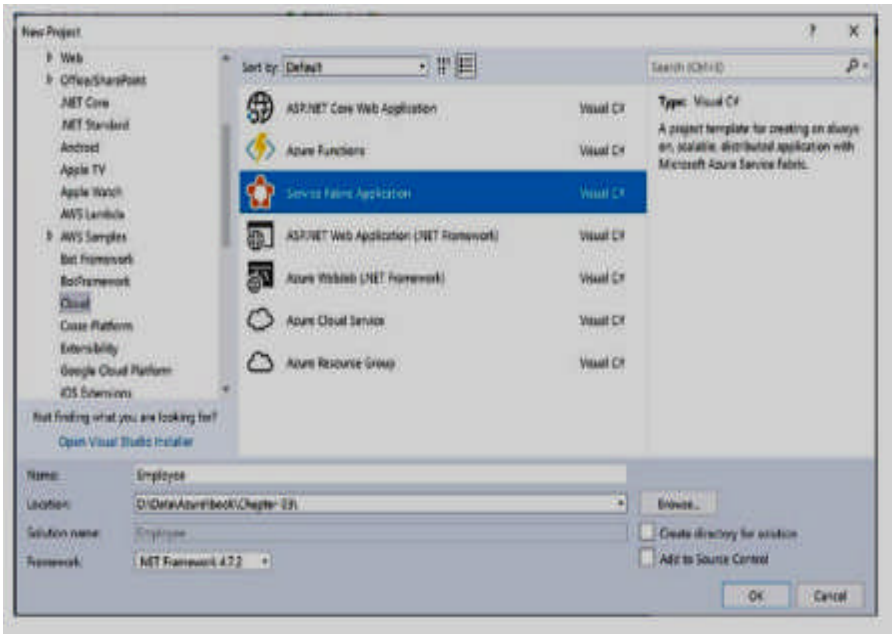


Figure: 2-8. New Service Fabric application [3]

5. Choose Stateful ASP.NET Core, as depicted in Figure 2-9

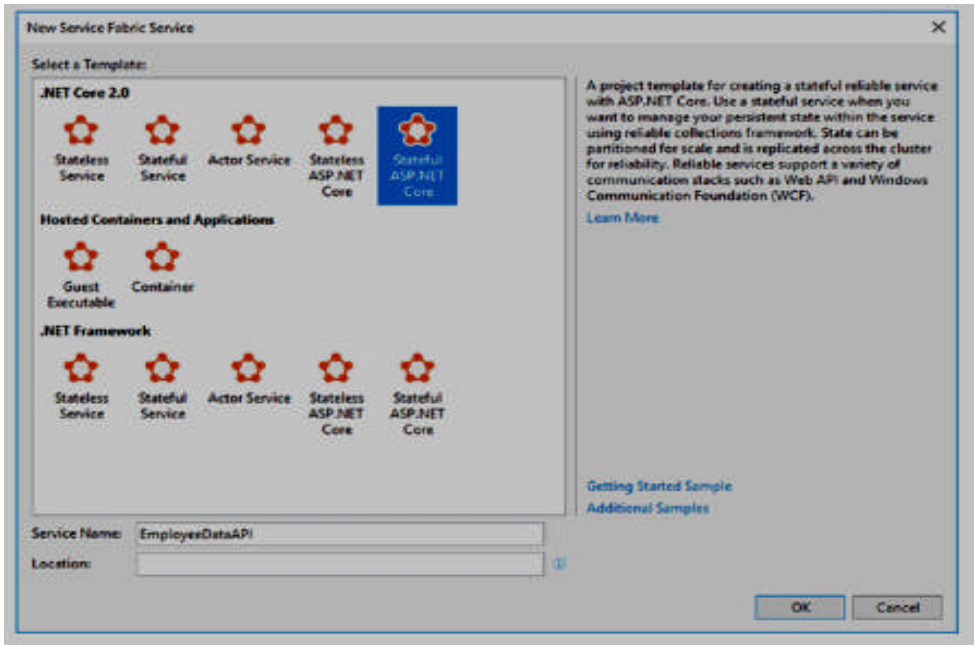


Figure:2-9. New Stateful ASP.NET Core API [3]

6. You see a screen that looks like Figure 2-10. Click OK

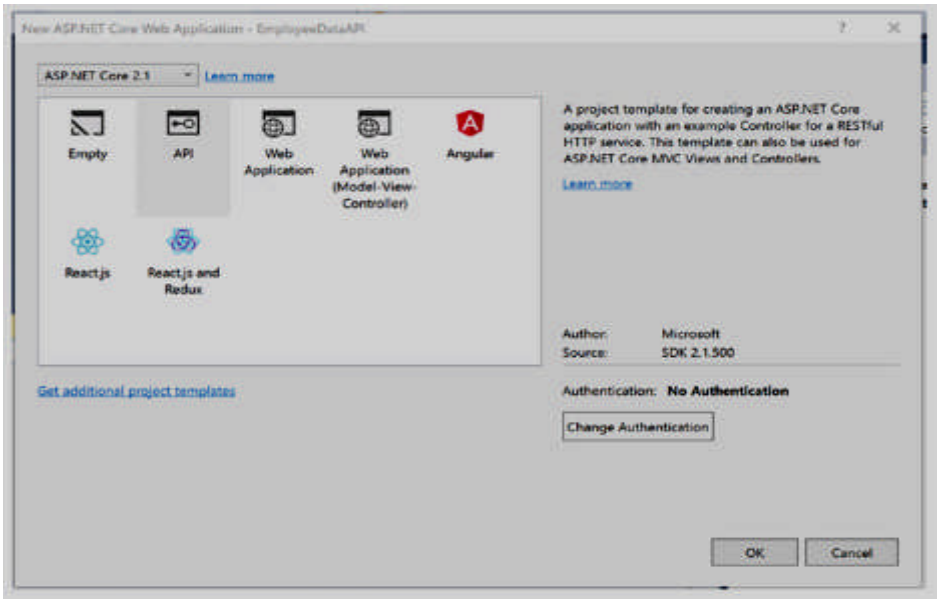


Figure: 2-10. Choose API using (ASP.NET Core 2.1) [3]

7. Right-click the Controller folder in the Employee Data API project and select Add ➤ New Controller, as shown in Figure 2-11. Select API Controller and name the controller Employee Controller

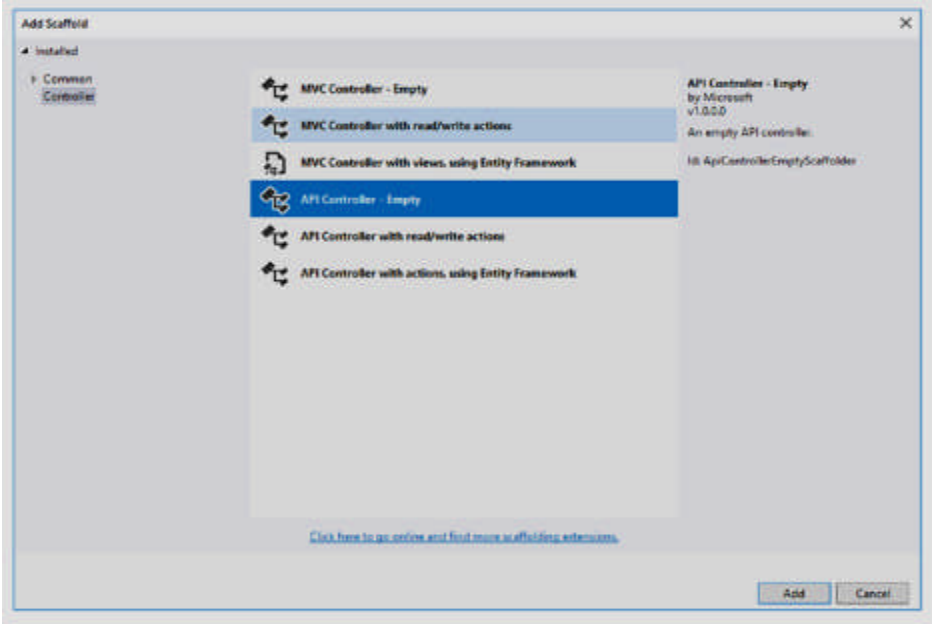


Figure:2-11. New API controller [3]

8. Make sure that the Nu Get packages shown in Figure 2-12 are installed.

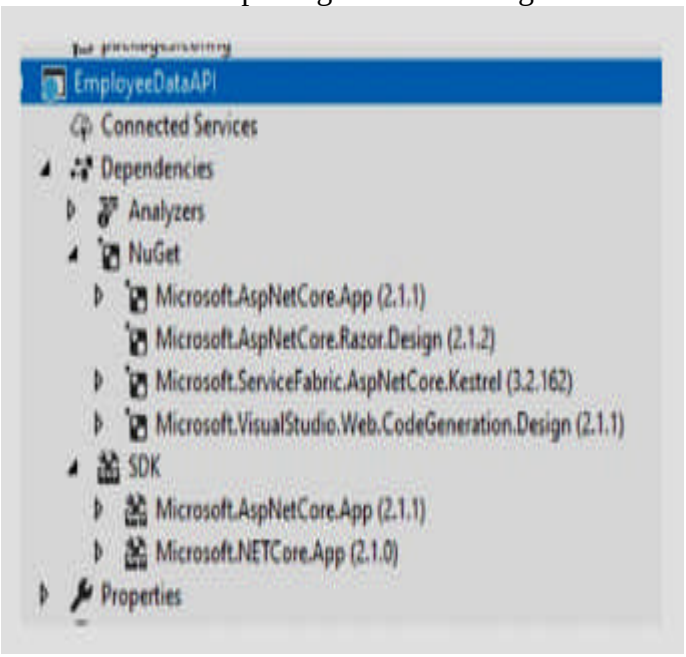


Figure: 2-12. NuGet Packages [3]

9. Replace the file content with the following and compile the changes[3].

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.ServiceFabric.Data;
 using Microsoft.ServiceFabric.Data.Collections;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace EmployeeDataAPI.Controllers
{
[Route("api/[controller]")]
[ApiController]
public class EmployeeController : ControllerBase
{
private readonly IReliableStateManager stateManager;
 public EmployeeController(IReliableStateManager stateManager)
{
 this.stateManager = stateManager;
 }
 [HttpGet]
public async Task<ActionResult<List>>GetAll()
{
 CancellationToken ct = new CancellationToken();
 IReliableDictionary employees = await
this.stateManager.GetOrAddAsync>("employees");
List employeesList = new List();
using (ITransaction tx = this.stateManager. CreateTransaction())
{
Microsoft.ServiceFabric.Data.IAsyncEnumerable<KeyValuePair> list =
await employees.CreateEnumerableAsync(tx);
Microsoft.ServiceFabric.Data.IAsyncEnumerator<KeyValuePair>
enumerator = list.GetAsyncEnumerator();
while (await enumerator.MoveNextAsync(ct))
{
employeesList.Add(enumerator.Current.Value);
}
 }
 return new ObjectResult(employeesList);
}
 [HttpGet("{id}")] public async Task<ActionResult>GetEmployee(string
id)
{
 IReliableDictionary employees = await
this.stateManager.GetOrAddAsync>("employees"); Employee employee =
null;
```

```
using (ITransaction tx = this.stateManager. CreateTransaction())
 {
ConditionalValue currentEmployee = await
employees.TryGetValueAsync(tx, id);
if (currentEmployee.HasValue)
{
employee = currentEmployee.Value;
 }
 }
return new OkObjectResult(employee);
}
[HttpPost]
public async Task Post(Employee employee)
{
IReliableDictionary employees = await
this.stateManager.GetOrAddAsync>("employees"); using (ITransaction tx
= this.stateManager. CreateTransaction())
{
 ConditionalValue currentEmployee = await
employees.TryGetValueAsync(tx, employee. Id.ToString());
 if (currentEmployee.HasValue)
 {
await employees.SetAsync(tx, employee. Id.ToString(), employee);
}
Else
 {
 await employees.AddAsync(tx, employee. Id.ToString(), employee);
}
 await tx.CommitAsync();
}
return new OkResult();
}
[HttpDelete("{id}")]
public async Task Delete(string id)
 {
```

```
IReliableDictionary employees = await
this.stateManager.GetOrAddAsync>("employees"); using (ITransaction tx
= this.stateManager. CreateTransaction())
{
 if (await employees.ContainsKeyAsync(tx, id))
{
 await employees.TryRemoveAsync(tx, id);
 await tx.CommitAsync();
 return new OkResult();
}
else
{
return new NotFoundResult();
}
}
}
 public class Employee
 {
public string Name{ get; set; }
public string Mobile { get; set; }
public long Id { get; set;}
 public string Designation { get; set; }
}
}
```

Service Fabric gives a substantial quantity in the form of reliable lines and a reliable dictionary. By utilizing these classes, Service Fabric secures the state is partitioned, replicated, and transacted in a partition. Moreover, all the operations in a reliable dictionary object need a transaction object. By default, the Visual Studio template utilizes range partitioning; these details can be seen in Application Manifest.xml, which remains in the Application Package Root folder of the Employee project. By default, the partition count is set to one. The replica count is set to three, which involves a copy of the service code, and data will be deployed on three nodes. Only one copy is active, named as the primary, and the other two are inactive and utilized only in case of failure.

Debugging the Application

By performing all the actions in the preceding segment, our growth is consummate. Here are the steps that debug the application to build an

employee recording in the Service Fabric reliable collection that uses the advanced web interface and data API.

1. Right-click the Employee project and set the Application URL to "http://localhost:19080/ Explorer". By default, Service Fabric Explorer runs on 19080. This assures the successful deployment of the service to a local cluster. It drives Service Fabric Explorer.
2. Make assured that the Employee project is established at startup.
3. Click F5. It uses our Service Fabric application to the local development cluster.
4. In Service Fabric Explorer, click Application. Click fabric://Employee, fabric://Employee/ Employe Web, Partition ID, and Node ID. Follow the value of the endpoint. (In our case, Employee Web is hosted at http://localhost:8780.)
5. You can also get the Employee Web port number from Service Manifest.xml.
6. Browse the http://localhost:8780/ URL to observe the web interface. Record employee information and click Create to create an employee record. The original data is protected by the Employee Data API in the Service Fabric's substantial collection, rather than an external database like Azure SQL

LET US SUM UP

Azure Service Fabric is a distributed systems platform that makes it simple to package, deploy, and manage scalable and reliable microservices and containers. We can build Service Fabric clusters everywhere, including Windows Server and Linux on-premises and other public clouds, along with Azure.

LIST OF REFERENCES

- https://cloudacademy.com/course/creating-an-app-on-azure-service-fabric/what-is-service-fabric/
- https://medium.com/@prasanna_vasan/azure-service-fabric-intro-41e1dc319a66
- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri
- https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview

BIBLIOGRAPHY

- https://cloudacademy.com/course/creating-an-app-on-azure-service-fabric/what-is-service-fabric/

- https://medium.com/@prasanna_vasan/azure-service-fabric-intro-41e1dc319a66
- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri
- https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview

UNIT END EXERCISES

1. Define Azure Service Fabric.
2. Describe the importance of application instances in Azure Service Fabric.
3. Define cluster in Azure Fabric Service
4. Define node in Azure Fabric Service
5. Define Service type in Azure Service Fabric.
6. Define Service instance in Azure Service Fabric.
7. Explain Micro Service
8. What are Micro services in Azure?
9. How does Azure service fabric work?
10. Describe different types of Micro Services.
11. Define Application type in Azure Service Fabric.
12. Brief about Programming models available in Azure Service Fabric.
13. What is a Container?
14. Explain the advantages of the container over virtual machines.
15. Which two metrics affect resource governance that is supported in Service Fabric?

MCQs

I. **Which service in Azure is used to manage resources in Azure?**
   A. Application Insights
   B. Azure Resource Manager
   C. Azure Portal
   D. Log Analytics

II. **A _____ role is a virtual machine instance running Microsoft IIS Web server that can accept and respond to HTTP or HTTPS requests.**
   A. Web
   B. Server
   C. Worker
   D. Client

❖❖❖❖

3

MONITORING AZURE SERVICE FABRIC CLUSTERS

Unit Structure :

3.0 OBJECTIVES

1) Azure Service Fabric is a distributed systems platform that creates it simple to package, use, and maintain scalable and reliable microservices and containers.
2) Service Fabric also marks the important challenges in developing and maintaining cloud-native applications.

3.1 INTRODUCTION

In any cloud environment, monitoring and diagnostics are crucial to testing, developing, and deploying workloads. For instance, we can trace how our applications are utilized, the activities conducted by the Service Fabric platform, our source utilization with performance counters, and the overall strength of the cluster. We can handle this information to diagnose and correct issues and restrict them from happening in the future.

Application monitoring

Application monitoring traces that how the characteristics and elements of our application are being utilized. We need to monitor our applications to make assured problems that affect users are found. The ability of application monitoring is on the users forming an application

and its services since it is unusual to the business logic of the application. Monitoring applications can be helpful in the following situations:

- How much traffic is my application experiencing? - Do we want to scale services to satisfy user requirements or approach a potential bottleneck in the application?
- Are my service to service calls successful and tracked?
- What actions are driven by the users of my application? - Collecting telemetry can manage coming feature growth and more reliable diagnostics for application errors
- Is my application driving unhandled exceptions?
- What is occurring within the services working inside my containers?

The excellent element of application monitoring is that developers can utilize whatever tools and framework they had like as it exists in the context of the application.

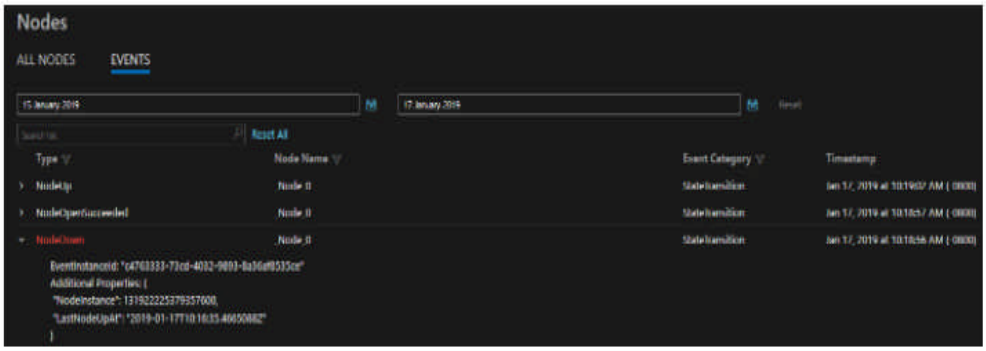
Platform (Cluster) monitoring

One of Service Fabric's aims is to have applications flexible to hardware crashes. This aim is obtained by the platform's system services' capacity to identify infrastructure problems and quickly failover workloads to other nodes in the cluster. But in this example, what if the system services themselves hold issues? Or if in trying to use or drive a workload, controls for the placement of services are violated? Service Fabric gives diagnostics for those and more to make assured that you are informed about exercise taking place in the cluster. Some example situations for cluster monitoring include:

Service Fabric gives a complete set of functions remarkable. These Service Fabric events can be obtained by the Event Store or the operational channel (event channel exposed by the platform).

Service Fabric event channels - On Windows, Service Fabric events are obtainable of a single ETW provider by a set of related log Level Keyword Filters utilized to choose among Operational and Data & Messaging channels - that is how we divide outgoing Service Fabric events to be clarified on as required. On Linux, Service Fabric events occur in LTTng and are placed into one Storage table, from wherever they can be clarified as wanted. Those channels contain curated, structured events that can be utilized to properly explain the state of the cluster. Diagnostics are allowed by default at the cluster production time, which generates an Azure Storage table wherever the events from those channels are posted for you to doubt in the future.

· EventStore - The EventStore is a characteristic given by the platform that produces Service Fabric platform events possible in the Service Fabric Explorer and through REST API. We can view a snapshot view of what is running on in the cluster for all entity e.g. node, service, application, and query based on the time of the event.



The diagnostics given are in the pattern of a complete set of events remarkable. Certain Service Fabric events represent activities performed by the platform on various entities such as Nodes, Applications, Services, Partitions, etc. In the current situation earlier, if a node were to go below, the platform would release a Node Down event that could be informed quickly by our monitoring tool of preference. Other typical cases involve Application Upgrade Rollback Started or Partition Reconfigured while a failover. Similar events are possible on both Windows and Linux clusters. The events are transmitted within standard channels on both Windows and Linux and can be read by any monitoring tool that supports these. The Azure Monitor solution is Azure Monitor logs. Azure Monitor logs integration involves a system operational dashboard for the cluster and some example questions which can generate alerts. More cluster monitoring concepts are possible at Platform level event and log generation.

Health monitoring

The Service Fabric platform adds a health model, which gives extensible health reporting for the status of entities in a cluster. Each node, application, service, partition, replica, or instance, has a continuously updatable health status. The health status can either be "OK", "Warning", or "Error". Study of Service Fabric events as verbs done by the cluster to different entities and health as an identifier for each entity. Every time the health of the appropriate entity transitions, an event will also be released. In this process, we can set up doubts and alerts for health events in the monitoring tool of preference, like any other event.

If the application is running by a grade and you have validation tests missing, we can write to Service Fabric Health utilizing the Health API to intimate the application is no healthier, and Service Fabric will automatically roll back the upgrade!

Watchdogs

Usually, a watchdog is a separate service that watches health and load over services, pings endpoints, and reports sudden health events in the cluster. It can support stop errors that may not identify based on the appearance of a single service. Watchdogs are also a great place to host code that executes corrective procedures that do not need user communication, such as washing up log files in storage at certain time pauses. If you require a completely executed, open-source SF watchdog service that involves a simple-to-use watchdog extensibility model and that operates in both Windows and Linux clusters. Fabric Observer is production-ready software. We help to use Fabric Observer to our test and production clusters and increase it to satisfy the requirements either by its plug-in model or by forking it and writing your built-in observers. The former (plug-ins) is the suggested approach.

Infrastructure (performance) monitoring

Monitoring the infrastructure is an important part of learning the state of the cluster and resource utilization. Measuring system execution depends on various circumstances that can be biased depending on the workloads. Those circumstances are typically covered by performance counters. These performance counters can arise from a variety of resources including the operating system, the .NET framework, or the Service Fabric platform itself. Some situations where they would be helpful are

- Am I using my hardware efficiently? Do you require to utilize your hardware at 90% CPU or 10% CPU? This arrives in handy when comparing the cluster, or optimizing the application's processes.
- Can I divine infrastructure problems proactively? - various issues are introduced by sudden changes (drops) in execution, so you can handle performance counters such as network I/O and CPU utilization to the divine and diagnose the issues proactively.

A record of performance counters that should be received at the infrastructure level can be seen at Performance metrics. Service Fabric also gives a set of performance counters for the Reliable Services and Actors programming models. If we are utilizing either of those models, those performance counters can report assuring that our actors are spinning up and down perfectly, or that our good service requests are being managed quickly enough.

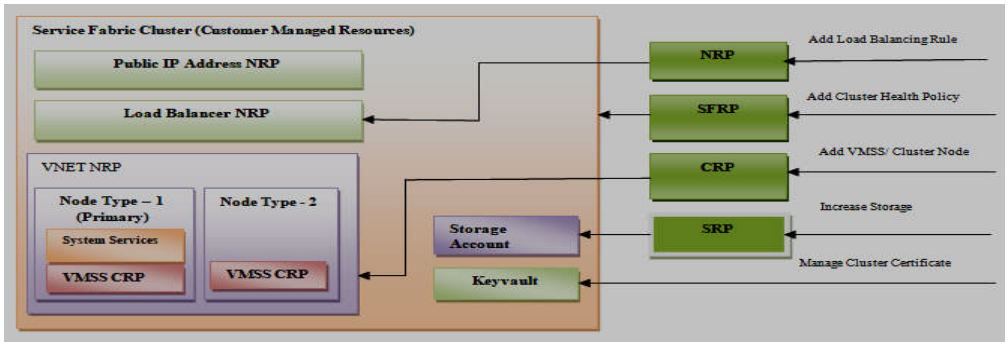
The Azure Monitor solution to get these is Azure Monitor logs just like platform level monitoring. We should accept the Log Analytics agent to get the proper performance counters and see them in Azure Monitor logs.

3.2 AN OVERVIEW

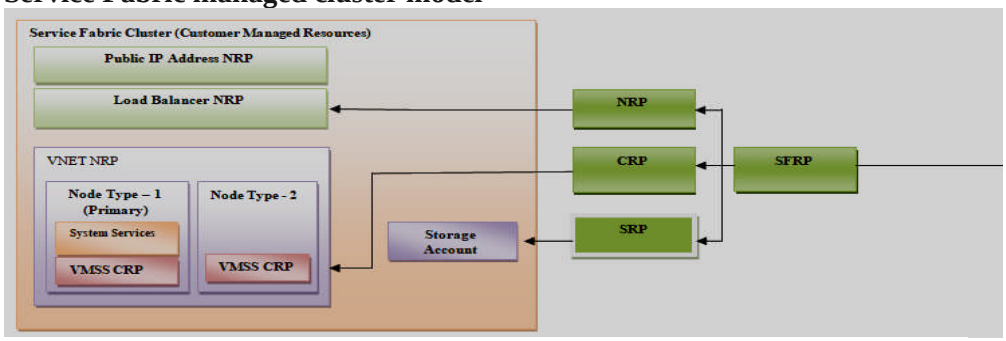
Service Fabric-managed clusters are a growth of the Azure Service Fabric cluster resource model that streamlines our deployment and cluster management practice.

The Azure Resource Model (ARM) template for regular Service Fabric clusters needs you to determine a cluster resource beside various helping resources, all of which must be "wired up" perfectly (upon deployment and during the lifecycle of the cluster) for the cluster and the services to operate correctly. In opposition, the encapsulation model for Service Fabric-managed clusters consists of a single, *Service Fabric-managed cluster* resource. All of the underlying resources for the cluster are separated away and maintained by Azure on your behalf.

Service Fabric traditional cluster model



Service Fabric managed cluster model



In terms of size and complexity, the ARM template for a Service Fabric managed cluster is about 100 lines of JSON, versus some 1000 lines needed to determine a distinctive Service Fabric cluster:

Service Fabric resources	Service Fabric managed cluster resources
Service Fabric cluster	Service Fabric managed cluster
Virtual machine scale set(s)	
Load balancer	
Public IP address	
Storage account(s)	
Virtual network	

Service Fabric managed clusters supply many advantages over traditional clusters:

Simplified cluster deployment and management

- Use and manage a single Azure resource
- Certificate management and autorotation
- Reduced scaling operations

Prevent operational errors

- Prevent configuration mismatches with underlying resources
- Block risky operations (such as removing a roots node)

Best habits by default

Simplified security and stability settings

There is no extra cost for Service Fabric-managed clusters exceeding the cost of underlying resources needed for the cluster, and the same Service Fabric SLA uses for managed clusters.

Service Fabric managed cluster SKUs

Service Fabric managed clusters are accessible in both Basic and Standard SKUs.

Feature	Basic	Standard
Network resource (SKU for Load Balancer, Public IP)	Basic	Standard
Min node (VM instance) count	3	5
Max node count per node type	100	100
Max node type count	1	20
Add/remove node types	No	Yes
Zone redundancy	No	Yes

Azure Application Insights

Application Insights is an extensible application performance management (APM) service for web developers on various platforms. It controls live web applications and automatically identifies performance irregularities. It also holds powerful analytics tools to support you diagnose problems and learning what users do with the application.

3.3 RESOURCE MANAGER TEMPLATE

Resource Manager Template

The Azure Resource Manager template enables us to use, monitor, and control resolution sources as a group on Azure. Here include three areas for Service Fabric monitoring as following.

- Application monitoring
- Cluster monitoring
- Infrastructure monitoring

Application Monitoring

Application monitoring explains the methods of the application's characteristics and elements, which supports determine their influence on the users. Application monitoring also informs debug and privilege logs, which are fundamental for diagnosing and solving a problem once the application is expanded. It is the duty of the developer's to add proper monitoring. You can utilize any modern instrumentation framework to add application monitoring, yet some of the recommended options are Application Insights SDK, Event Source, and ASP.NET Core Logging Framework.

3.4 ADDING APPLICATION MONITORING TO A STATELESS SERVICE USING APPLICATION INSIGHTS

Computing Application Monitoring to a Stateless Service Utilizing Application Insights We will produce an easy ASP.NET MVC–based API to handle employees. In this instance, we will save the employee data in an Azure SQL database rather than a good collection so that we can explain how to control query messages in Azure Application Insights. To display the monitoring of a REST API call, we are creating a call to the Translator Text API in Azure to transliterate the first name of an employee in the Hindi (Devanagari) script. We can restore the call with any other REST call, as the intention here is to display the monitoring of remote calls in Azure Application Insights. In Azure subscription, we can generate a Translator Text API using the Free tier to perform this unit

Setting up the Development Environment

Let's set up.

1. Install Visual Studio 2017.
2. Install the Microsoft Azure Service Fabric SDK.
3. Build the Translator Text API in Azure subscription and create a record of the access key.
4. Generate a blank Azure SQL Database and hold the connection string with SQL Authentication helpful.
5. Make assured that the Service Fabric local cluster on Windows is working.
6. Make assured that the Service Fabric Azure cluster on Windows is working.

Create an ASP.NET Core Web AP

Now let's begin the API.

1. Start Visual Studio 2017 as an administrator.
2. Design a project by selecting File ➢New ➢Project.
3. In the New Project dialog, choose Cloud ➢Service Fabric Application.
4. Name the Service Fabric application Employee App (as seen in Figure 3-1) and click OK

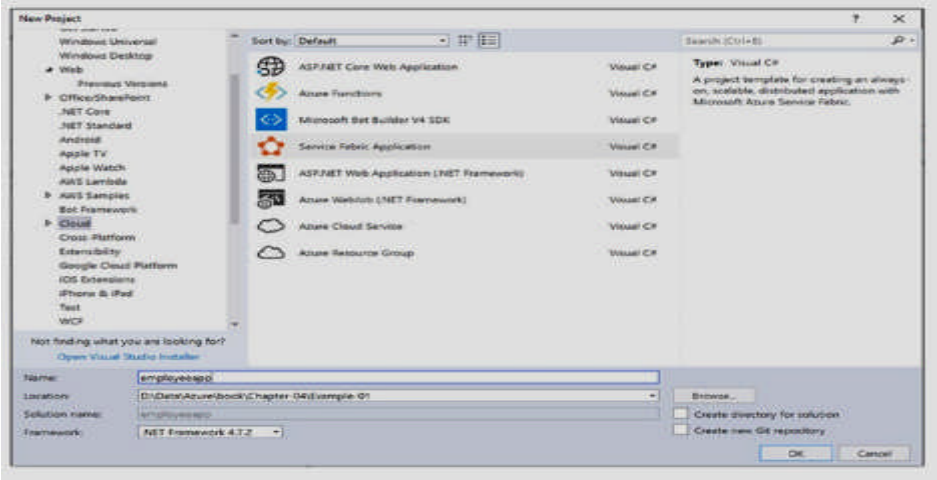


Figure: 3-1. Create Service Fabric application [3]

5. Name the stateless ASP.NET Core service Employee. Stateless.Api (as seen in Figure 3-2) and click OK.

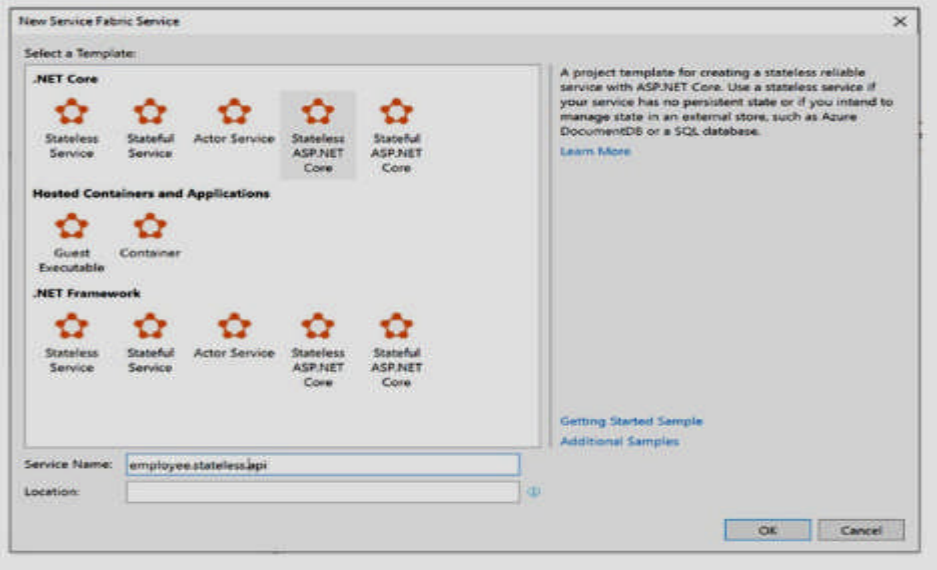


Figure:3-2. Stateless ASP.NET Core [3]

6. Choose the API and click OK. Make sure that ASP.NET Core 2.2 is selected, as shown in Figure 3-3.

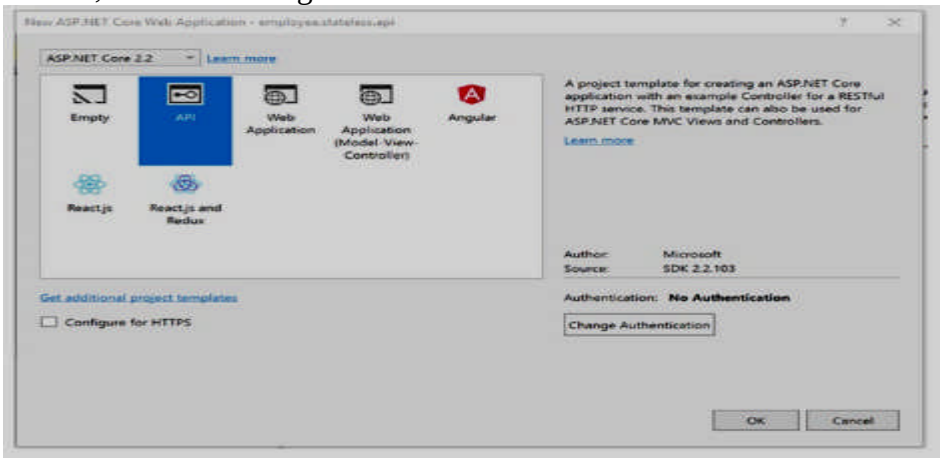


Figure: 3-3. API with ASP.NET Core 2.2[3]

7. Right-click the employee.stateless.api project and select Add ➤ Connected Service, as seen in Figure 3-4

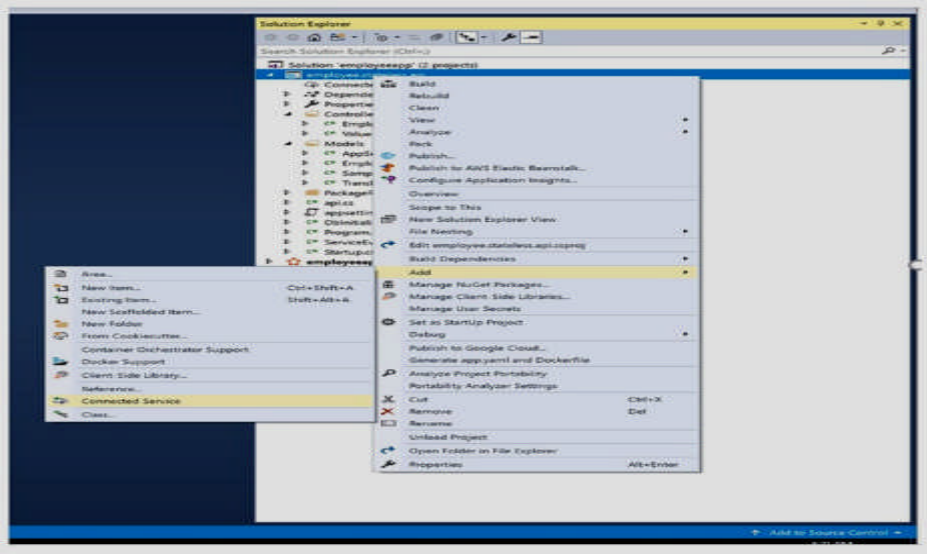


Figure: 3-4. Add connected service [3]

8. Choose Monitoring with Application Insights, as seen in Figure 3-5.

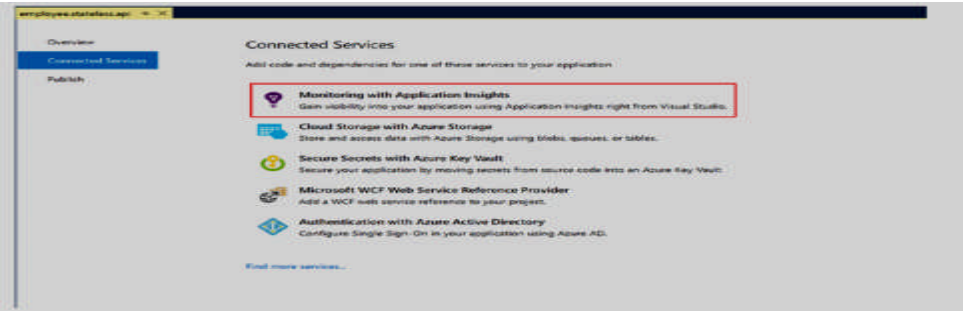


Figure 3-5. Monitoring with Application Insights [3]

9. Click Get Started, as seen in Figure 3-6

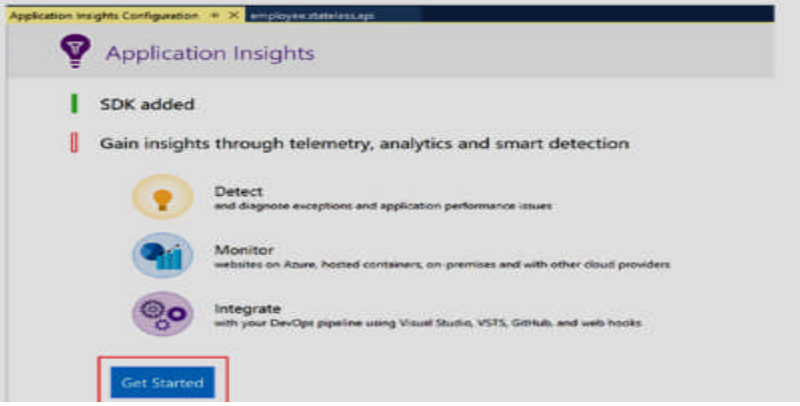


Figure: 3-6. Get started [3]

10. Choose the right Azure subscription and Application Insights resource. Once done, click Register, as seen in Figure 3-7

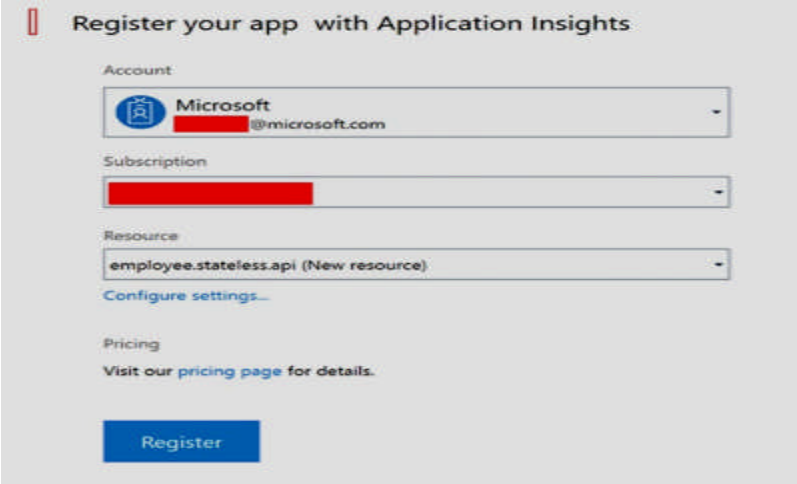


Figure 3-7. Choose Azure subscription [3]

It takes a few minutes to create the Application Insights resource in your Azure subscription. During the registration process, you see the screen shown in Figure 3-8

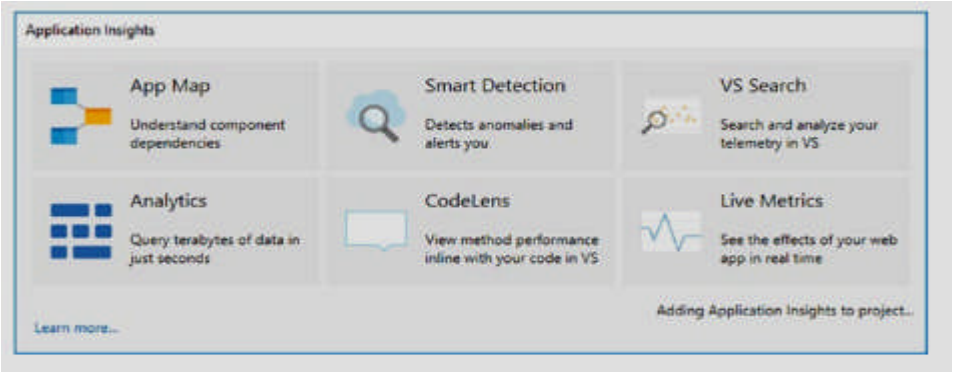


Figure 3-8. Registration process [3]

11. Once the Application Insights configuration is complete, you see the status as 100%. If you see the Add SDK button (as shown in Figure 3-9), click it to achieve 100% status, as seen in Figure 3-10.

Figure: 3-9. Add SDK [3]

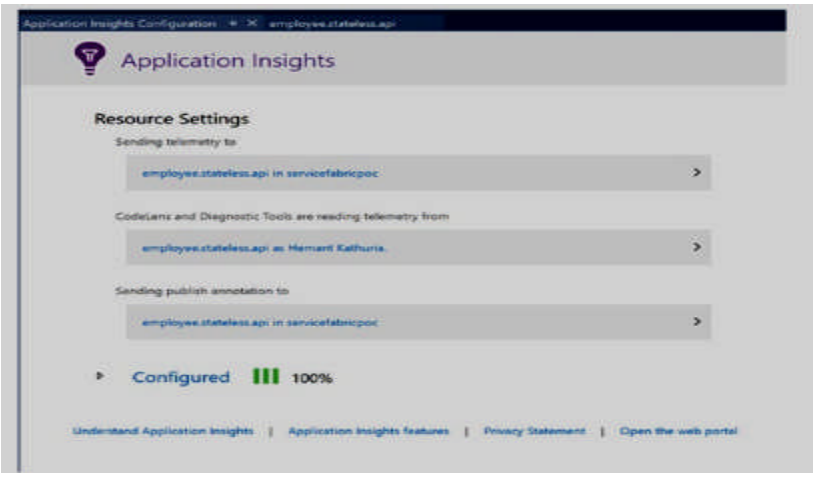


Figure: 3-10. Application Insights SDK installation complete [3]

12. To approve the Application Insights configuration, check the instrumentation key in appsettings.json.
13. Right-click the employee.stateless.api project to add provinces for the following NuGet packages.
    a. Microsoft.EntityFrameworkCore.SqlServer
    b. Microsoft.ApplicationInsights.ServiceFabric.Native
    c. Microsoft.ApplicationInsights.AspNetCore

We have done with the configuration. Now let's join Employee Controller, which is effective for implementing CRUD operations on Azure SQL Database.

1. Right-click the employee.stateless.api project and attach a folder called Models. Attach the following classes from the resources folder. a. AppSettings.cs b. Employee.cs c. Sample Context.cs d. Translation Response.cs
2. Right-click the employee.stateless.api project and attach a file named DbInitializer.cs. Substitute that content with the following content.

```
using employee.stateless.api.Models;
 namespace employee.stateless.api
{
 /// /// Class to initialize database ///
 public class DbInitializer
{
private SampleContext _context = null;
 public DbInitializer (Sample Context context)
{
_context = context;
}
```

```
public void Initialize()
{
_context.Database.EnsureCreated();
}
 }
 }
```

3. Open Api.cs and replace the contents of the Create Service Instance Listeners method with the following content.

```
return new Service Instance Listener[]
{
new Service Instance Listener(service Context => new Kestrel Communication Listener(service Context, "Service Endpoint", (url, listener) =>
{
Service Event Source.Current. Service Message (service Context, $"Starting Kestrel on {url}"); return new Web Host Builder() .Use Kestrel()
//Add the below code to
 read appsettings.json .Configure App Configuration ( (builder Context, config) =>
{
 config.AddJsonFile ("appsettings.json", optional: false, reloadOnChange: true);
}) .Configure Services( services => services .AddSingleton (service Context)
//Make sure the below line exists for application insights integration .Add Singleton ((service Provider) => Fabric Telemetry Initializer Extension. Create Fabric Telemetry Initializer (service Context)))
.Use Content Root (Directory. Get Current Directory()) .Use Startup()
.Use Application Insights() .Use Service Fabric Integration (listener, Service Fabric Integration Options.None) .Use Urls(url) .Build();
}))
 };
```

Make sure that you have the following namespaces imported on top of the Api.cs file.

```
using System.Collections.Generic;
using System.Fabric;
using System.IO;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
 using Microsoft.ServiceFabric.Services.Communication. AspNetCore;
using Microsoft.ServiceFabric.Services.Communication.Runtime;
using Microsoft.ServiceFabric.Services.Runtime;
using Microsoft.Extensions.Configuration;
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.ServiceFabric;
```

4. Open Startup.cs and replace the contents of the Configure Services method with the following content.
services.Add Db Context(options => //registring the use of SQL server options.Use Sql Serve r(Configuration.Get Connection String ("Default Connection"))); services.AddSingleton();
services.AddHttpClient();
services.Configure(Configuration. GetSection("AppSettings"));
var serviceProvider = services.BuildServiceProvider();
var dbInitializer = serviceProvider.GetRequiredService ();
dbInitializer.Initialize();
services.Add Mvc().Set Compatibility Version (Compatibility Version. Version_2_2);
5. Right-click the controller folder in the employee. stateless.api project and add a controller called Employee Controller.cs. Replace that content with the following content.
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using employee.stateless.api.Models;
using Microsoft.AspNet Core.Mvc;
using Microsoft.Entity FrameworkCore;
using Microsoft.Extensions.Options;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
namespace employee.stateless.api.Controllers
{
[Route("api/[controller]")] [Api Controller] public class EmployeeController :
ControllerBase { /// /// Context ///[3]

3.5 CLUSTER MONITORING

Cluster Monitoring

One of the notable characteristics of an Azure Service Fabric cluster is producing applications flexible to hardware crashes. For instance, if Service Fabric system services are becoming issues in using workloads, or services not capable to support placement laws, Service Fabric provides diagnostic logs to control those situations. Service Fabric shows many structured platform events for effective diagnosis. On Azure, for windows clusters, it's recommended to utilize Diagnostic Agents and Azure Monitor Logs. Azure Monitor Logs is also recommended for Linux workloads but among various configurations.

Diagnostic Agents

The Windows Azure Diagnostic expansion enables you to assemble all the logs from all the cluster nodes to a middle location. The prime location can be Azure Storage, and it can give the logs to Azure Application Insights or Event Hubs. Diagnostic agents can be used during

the Azure portal when generating a Service Fabric cluster. We can also utilize the Resource Manager template to add a diagnostic agent to a current Service Fabric cluster if it was not attached when building the cluster. Figure 3-20 shows the diagnostics options when creating a Service Fabric cluster.

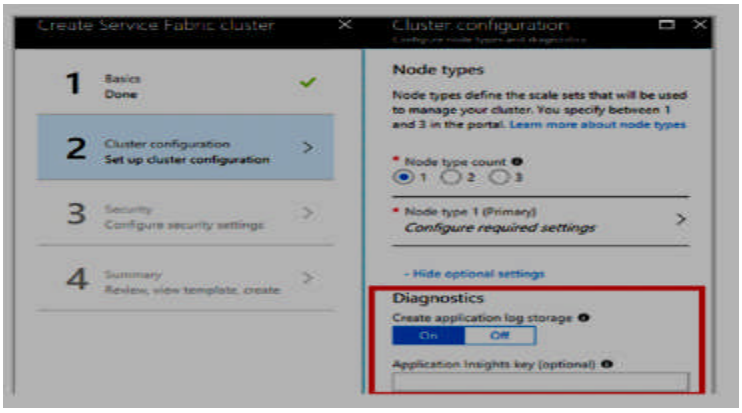


Figure: 3-20. Configure diagnostics agent [3]

Azure Monitor Logs

Microsoft supports utilizing Azure Monitor Logs to control cluster-level events in a Service Fabric cluster. To manage the decision, the diagnostic logs for the Service Fabric cluster must be allowed. Setting up Azure Monitor Logs is achieved through Azure Resource Manager, Power Shell, or Azure Marketplace. Here we support the Azure Marketplace route because it is user-friendly and simple to learn.

1. Select New in the left navigation menu of the Azure portal.
2. Search for Service Fabric Analytics. Select the resource that appears.
3. Select Create, as seen in Figure 3-21

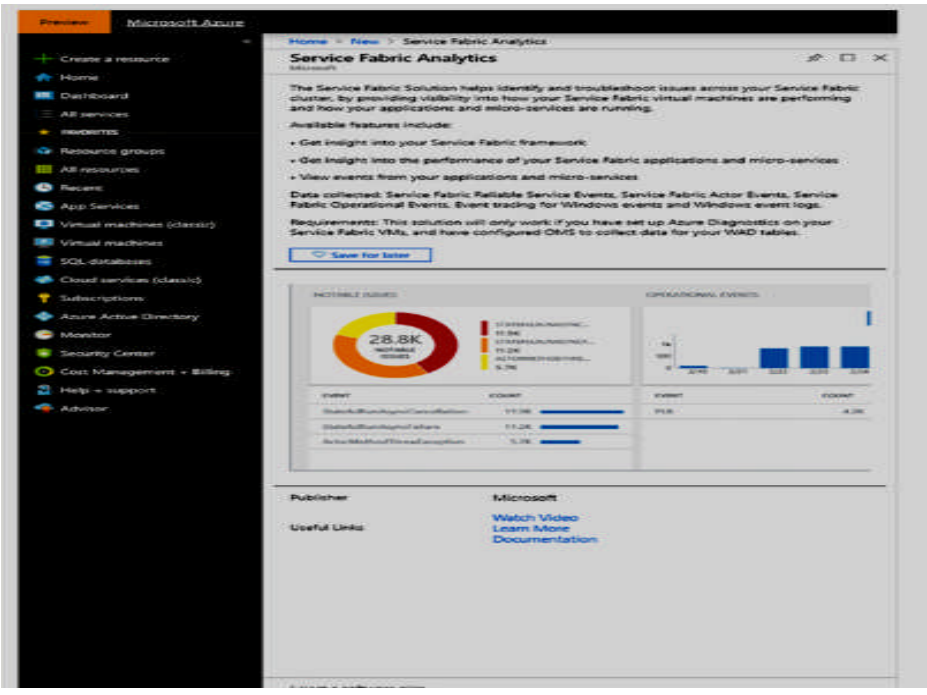


Figure: 3-21. Create Service Fabric analytics [3]

1. Make a new Log Analytics workspace, as seen in Figure 3-22. Once it is created, you requirejoining it to your Azure Service Fabric cluster

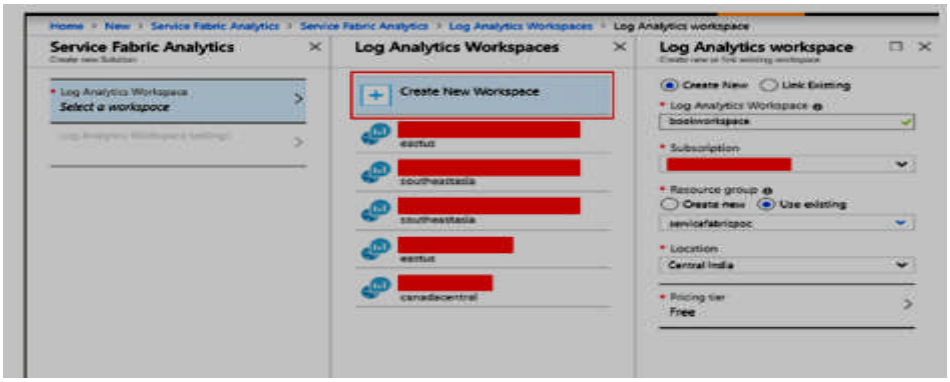


Figure: 3-22. Create log analytics [3]

2. Go to the source group where you built the Service Fabric analytics solution. Select Service Fabric and go to its Overview page.
3. Choose Storage Account Logs under the Workspace Data Sources option.
4. Click Add, as seen in Figure 3-23

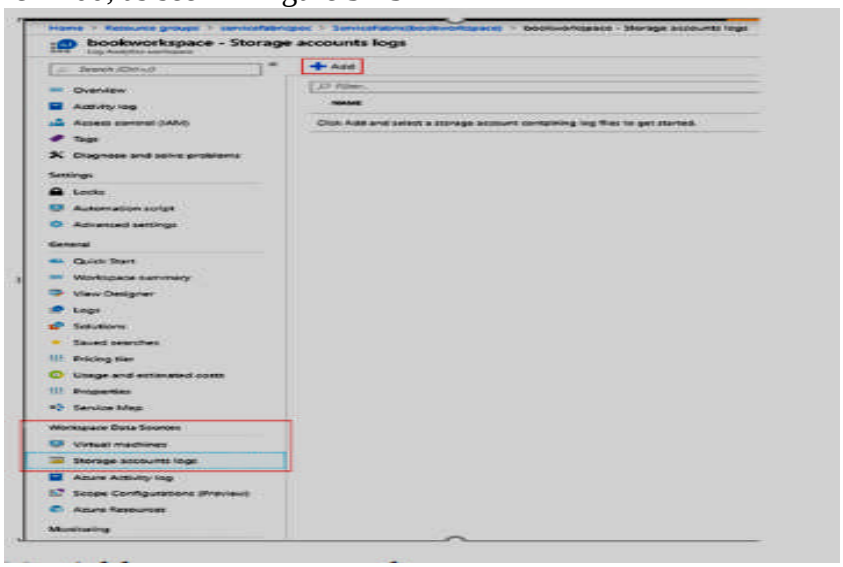


Figure: 3-23. Add storage account logs [3]

5. Select the storage account generated with the Service Fabric cluster. The default name for the Service Fabric cluster storage account begins with sfdg.
6. Choose Service Fabric Events as the data type.
7. Make assured that the resource is set to WAD Service Fabric *Event Table, as seen in Figure 3-24

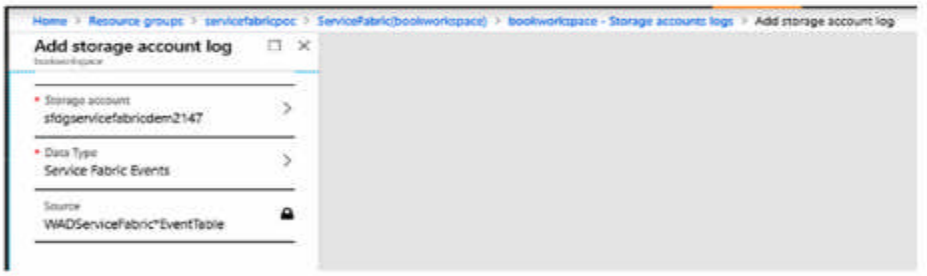


Figure 3-24. WAD Service Fabric *Event Table [3]

Once completed, on the Overview page, we can see a summary of Service Fabric events. Please remark that it may use 10 to 15 minutes for data to arrive in this view, as seen in Figure 3-25

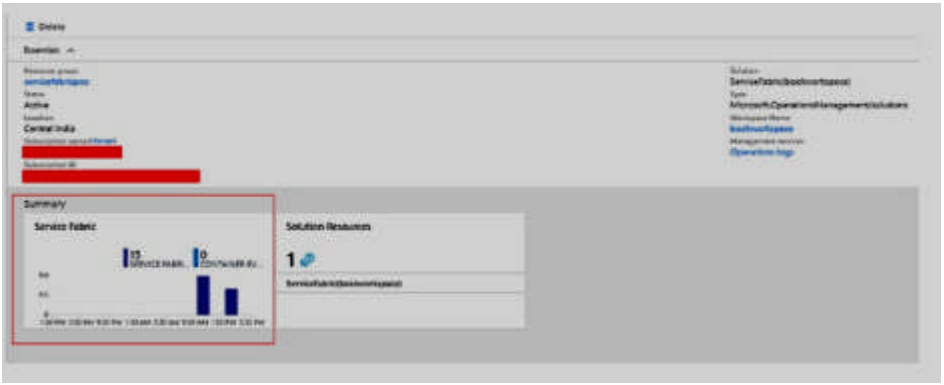


Figure 3-25. Overview of Service Fabric events [3]

8. Click the Service Fabric tile to see more reported information about the cluster events, as seen in Figure 3-26

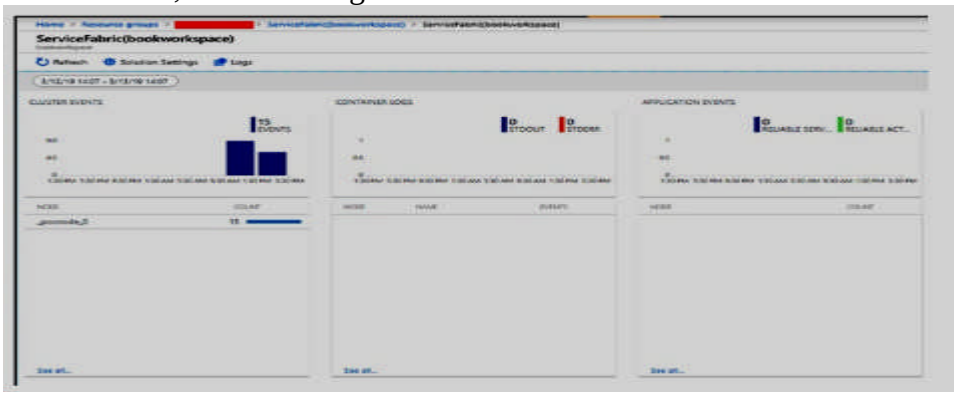


Figure 3-26. Cluster event details [3]

3.6 INFRASTRUCTURE MONITORING

Infrastructure Monitoring

Azure Monitor Logs is approved for controlling support parameters such as CPU utilization, .NET performance counters, and Service Fabric performance counters (e.g., the number of objections from a secure service). To make the infrastructure logs, you are needed to attach a Log Analytics agent as a virtual machine scale set extension to the Azure Service Fabric cluster. Observe those actions to do this.

1. Go to the source group in which you built the Service Fabric Analytics resolution. Select Service Fabric and go to its Overview page. Select Log Analytics Workspace and click Advanced Settings under Settings.

2. Select Windows Servers. Create a note of the workspace ID and primary key, as seen in Figure 3-27

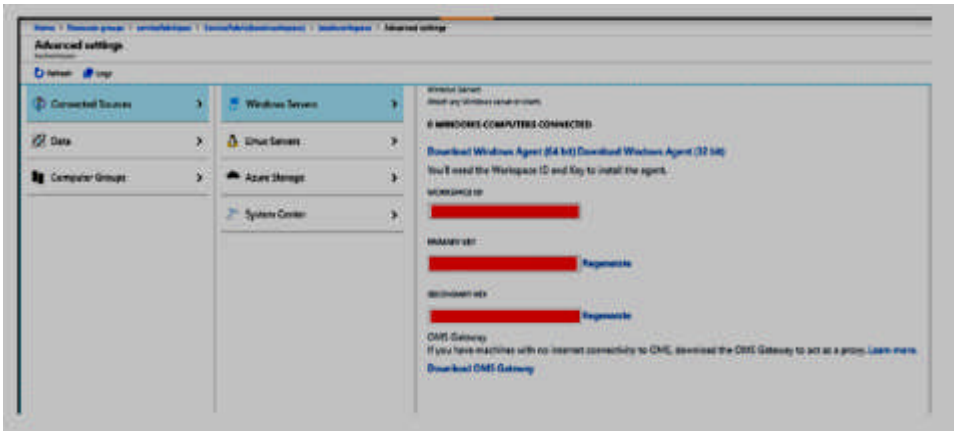


Figure 3-27. Windows Server details [3]

3. Open the Cloud Shell from the Azure portal to execute the command in the next step. The alternative is possible in the top-right corner of the Azure portal, as seen in Figure 3-28

Figure 3-28. Cloud shell [3]

4. Execute the following command to add the monitoring agent.
   az vmss extension set --name Microsoft Monitoring Agent --publisher Microsoft. Enterprise Cloud. Monitoring --resourcegroup --vmss-name --settings "{'workspaceId':"}" --protected-settings "{'workspace Key':"}"
5. Replace the workspace ID and workspace key obtained from the preceding step. Name Of Node Type is the name of the virtual machine system set resource that was automatically generated with your Service Fabric cluster. This command needs about 15 minutes to attach the log analytics agents on all the scale set nodes.
6. Go to the source group in which you built the Service Fabric Analytics solution. Choose Service Fabric and go to its Overview page. Choose Log Analytics Workspace and click Advanced Settings under Settings.
7. Select Data and Windows Execution Counters. Click Add the chosen performance counters. (For this sample exercise, we select the default performance counters, but you can select custom performance counters in real-world applications, as seen in Figure 3-29.)

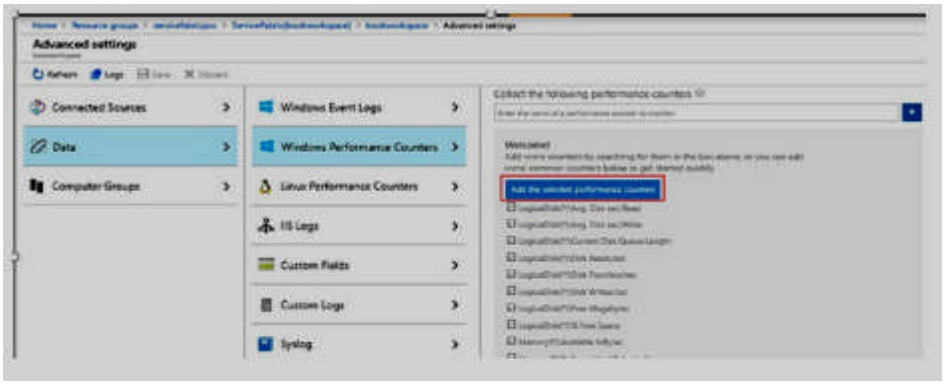


Figure 3-29. Windows performance counters [3]

8. Click Save.
9. Go to the source group in which you built the Service Fabric Analytics solution. Choose Service Fabric and go to its Overview page. Click the title for the Summary of Service Fabric events.
10. You see data for the chosen performance counters, like disk usage (MB). Click the chart to get more information, as seen in Figure 3-30. Please remark that it needs time to return the data in this section.

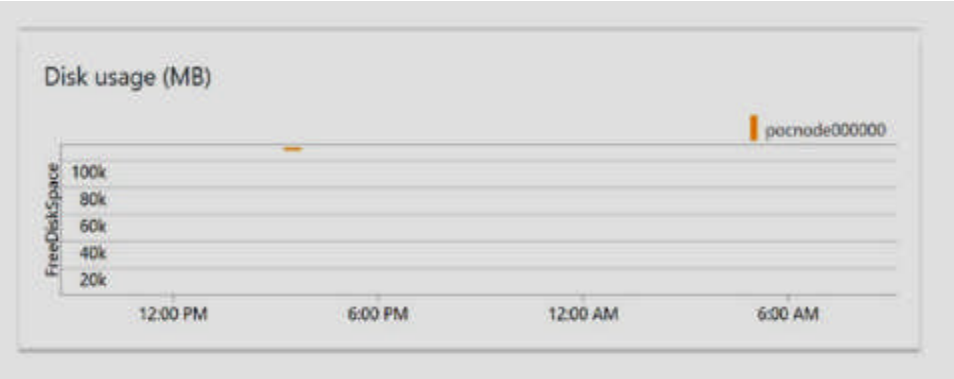


Figure 3-30. Disk usage details [3]

LET US SUM UP

Here we studied how to control a Service Fabric cluster and the applications used on it. We incorporated application monitoring, cluster monitoring, and infrastructure monitoring. Application Insights is a very efficient method of monitoring used applications because it efficiently controls remote HTTP and database calls with no extra energy. We also incorporated how you can use Diagnostic Agents and Azure Monitor Logs to control a Service Fabric cluster and infrastructure

LIST OF REFERENCES

- https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-diagnostics-overview
- https://docs.microsoft.com/en-us/azure/service-fabric/overview-managed-cluster

- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri

BIBLIOGRAPHY

- https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-diagnostics-overview
- https://docs.microsoft.com/en-us/azure/service-fabric/overview-managed-cluster
- Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring — Harsh Chawla Hemant Kathuri

UNIT END EXERCISES

1. Define Service Fabric Cluster.
2. Define cluster in Azure Fabric Service
3. Does a cluster have any minimum size limit? If yes, what and why?
4. How many nodes can be maintained on a service fabric cluster?
5. What is Azure Service Fabric cluster?
6. What is the maximum limit of Azure service fabric cluster in test environment?
7. How do you set up a service in fabric local cluster?
8. Which is the most important application of Microsoft Azure?
9. What are Azure applications?
10. What does a resource manager template consist of?
11. What is the purpose of Azure Resource Manager template?

Unit II

4

AZURE KUBERNETES SERVICE (AKS) AND MONITORING AKS

Unit Structure :

4.0 OBJECTIVES

After going through this unit, you will be able to:

- Learn to Deploy applications on AKS.
- Learn to monitoring AKS clusters

4.1 INTRODUCTION

4.1.1 What is Kubernetes

Kubernetes is a portable, extensible, open source orchestrator for deploying containerized applications. It was originally developed by Google, inspired by a decade of experience deploying scalable, reliable systems in containers via application-oriented APIs. Since its introduction in 2014, Kubernetes has grown to be one of the largest and most popular open source projects in the world. It has become the standard API for building cloud-native applications, present in nearly every public cloud. Kubernetes is a proven infrastructure for distributed systems that is suitable for cloud-native developers of all scales, from a cluster of Raspberry Pi computers to a warehouse full of the latest machines. It

provides the software necessary to successfully build and deploy reliable, scalable distributed systems.

Let's take a look at why Kubernetes is so useful by going back in time. There are various deployment models that can be used to deploy a software. These include Bare Metal, Virtual Machines, Container-based Infrastructure and Serverless Computing.

Bare Metal

- A bare metal deployment is essentially deploying to an actual computer. It is used to install a software directly on the target computer as shown in figure 4.1.
- In this method, software can directly access the operating system and the hardware.
- It is useful for situations requiring access to specialized hardware, or for High Performance Computing (HPC) applications.
- It is now used as infrastructure to host virtualization and cloud frameworks.

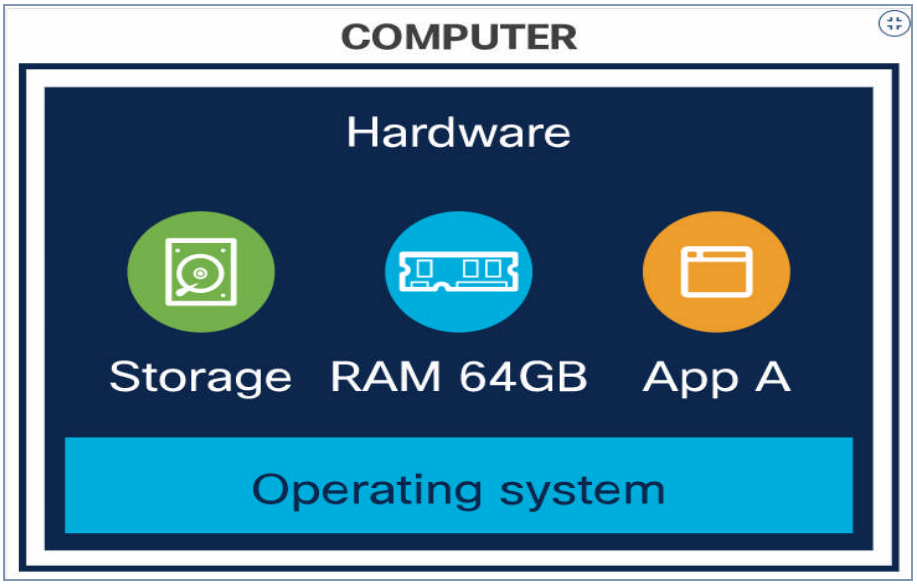


Figure 4.1 Bare Metal

Virtual Machines (VMs)

- Virtual machines share the resources of the host as Show in figure 4.2. It is like a computer within the computer and has its own computing power, network interfaces, and storage.
- Hypervisor is software that creates and manages VMs.
- VMs run on top of a hypervisor that provides VMs with simulated hardware, or with controlled access to underlying physical hardware.

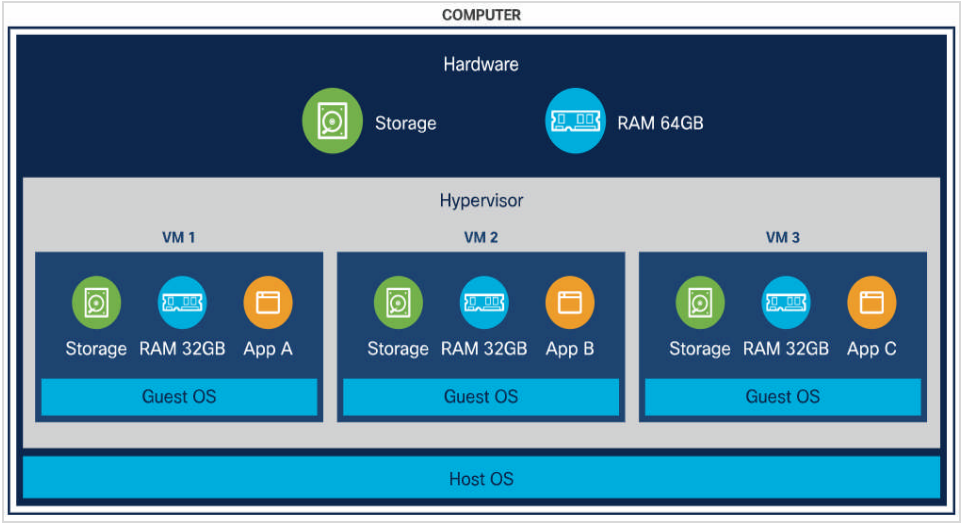


Figure 4.2 Virtual Machines

Container-based infrastructure

- Containers were designed to provide the same benefits as VMs, such as workload isolation and the ability to run multiple workloads on a single machine but are designed to start up quickly as shown in figure 4.3.
- Containers share resources of the host including the kernel.
- A container shares the operating system of the host machine and uses container-specific binaries and libraries.

Containers have become popular because they provide extra benefits, such as:

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability: not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be

deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.

- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

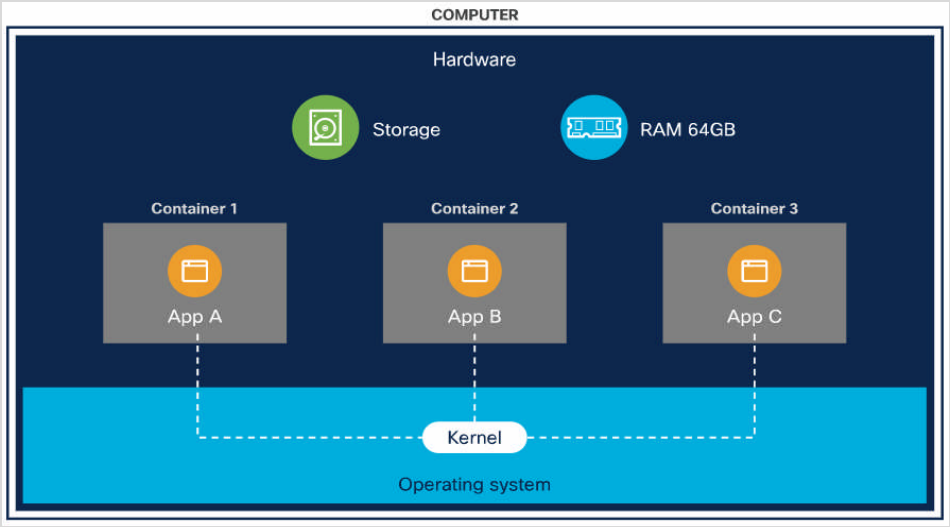


Figure 4.3 Container-based infrastructure

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behaviour was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

4.1.2 What Kubernetes provides you with:

- **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes

how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

- **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

4.2 MICROSOFT AZURE KUBERNETES SERVICES (AKS):

AKS is an open-source fully managed container orchestration service and is available on the Microsoft Azure public cloud that can be used to deploy, scale, and manage Docker containers and container-based applications in a cluster environment.

Microsoft Azure Kubernetes Service offers to provision, scaling, and upgrades of resources as per requirement or demand without any downtime in the Kubernetes cluster and the best thing about AKS is that you do not require deep knowledge and expertise in container orchestration to manage AKS.

AKS is certainly an ideal platform for developers to develop their modern applications using Kubernetes on the Azure architecture where Azure Container Instances are the pretty right choice to deploy containers on the public cloud. The Azure Container Instances help in reducing the stress on developers to deploy and run their applications on Kubernetes architecture.

So why use Azure AKS solutions:

- Hosts your Kubernetes environment.
- Easy integration with Azure services such as Load balancing, Azure Blob Storage, Azure Active Directory, Application Gateway, Azure Traffic Manager etc.
- Quick and easy to deploy.
- Hosted control plane.
- Easy and secure containerized applications management.
- Continuous Integration by adopting Azure Pipeline concept for Docker images creation for faster deployments and reliability.
- Create resources and infrastructure inside the Azure Kubernetes cluster through Deployments and services manifest files.

- AKS management service is free of charge in Microsoft Azure.

4.3 AZURE KUBERNETES SERVICE FEATURES:

- Microsoft Azure offers **Azure Kubernetes Service** that simplifies managed Kubernetes cluster deployment in the public cloud environment and manages health and monitoring of managed Kubernetes service.
- Customers can create AKS clusters using the Azure portal or Azure CLI and can manage the agent nodes.
- Some additional features such as advanced networking, monitoring, and Azure AD integration can also be configured.
- Features that Azure Kubernetes Service (AKS) offers are as follows:

Nodes and clusters:

In AKS, apps, and support, services are run on Kubernetes nodes and the AKS cluster is a combination of one or more than one node. And these AKS nodes are run on Azure Virtual Machines. Nodes that are configured with the same configuration are grouped together called node pools. Nodes in the Kubernetes cluster are scaled-up and scaled-down according to the resources are required in the cluster. So, nodes, clusters, and node pools are the most prominent components of your Azure Kubernetes environment.

Role-based access control (RBAC):

AKS easily integrates with Azure Active Directory (AD) to provide role-based access, security, and monitoring of Kubernetes architecture based on identity and group membership. You can also monitor the performance of your AKS and the apps.

Integration of development tools:

Another important feature of AKS is the development tools such as Helm and Draft are seamlessly integrated with AKS where Azure Dev Spaces can provide a quicker and iterative Kubernetes development experience to the developers. Containers can be run and debugged directly in the Azure Kubernetes environment with less stress on the configuration.

AKS also offers support for Docker image format and can also integrate with Azure Container Registry (ACR) to provide private storage for Docker images. And regular compliance with the industry standards such as System and Organization Controls (SOC), Payment Card Industry Data Security Standard (PCI DSS), Health Insurance Portability and Accountability Act (HIPAA), and ISO make AKS more reliable across the various business.

Running any workload in Microsoft Azure Kubernetes Services:

You can orchestrate any type of workload running in the AKS environment. You can move .NET apps to Windows Server containers,

modernize Java apps in Linux containers, or run microservices in Azure Kubernetes Service. AKS will run any type of workload in the cluster environment.

Removes complexities:

AKS removes your implementation, installation, maintenance, and security complexities in Azure cloud architecture. It also reduces substantial costs where no per-cluster charges are being imposed on you.

4.4 ADVANTAGES OF AKS:

- Deploy your applications quickly and predictably and easily coordinate deployments of your system.
- Constraint communications between containers.
- Continuously monitors and manages your containers.
- Improves reliability and availability.
- Scales your application to handle changes in load on the fly as needed.
- Better use of infrastructure resources.
- Cybersecurity is one of the most important aspects of modern applications and businesses. AKS integrates with Azure Active Directory (AD) and offers on-demand access to the users to greatly reduce threats and risks.
- AKS is also completely compliant with the standards and regulatory requirements such as System and Organization Controls (SOC), HIPAA, ISO, and PCI DSS.

4.5 KUBERNETES COMPONENTS

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.
- This document outlines the various components you need to have a complete and working Kubernetes cluster.
- Here's the figure 4.1 of a Kubernetes cluster with all the components tied together.

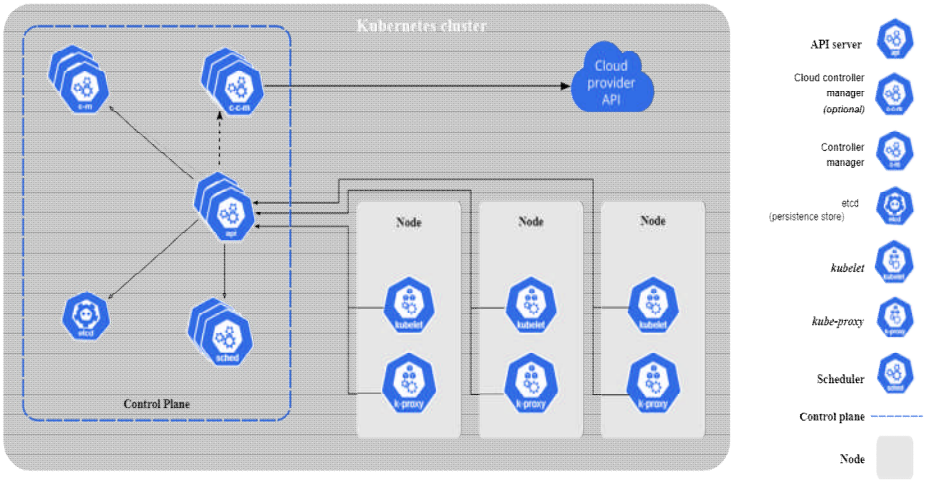


Figure 4.1 Kubernetes cluster

Control Plane Components

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Creating Highly Available clusters with kubeadm for an example control plane setup that runs across multiple VMs.

kube-apiserver

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

etcd

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for those data.

You can find in-depth information about etcd in the official documentation.

kube-scheduler

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

kube-controller-manager

Control plane component that runs controller processes.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

- Node controller: Responsible for noticing and responding when nodes go down.
- Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
- Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
- Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

cloud-controller-manager

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process. You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

- Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route controller: For setting up routes in the underlying cloud infrastructure
- Service controller: For creating, updating and deleting cloud provider load balancers

Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

Container runtime

The container runtime is the software that is responsible for running containers.

Kubernetes supports several container runtimes : Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

Addons

Addons use Kubernetes resources (Daemon Set, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

DNS

While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

Web UI (Dashboard)

Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

Container Resource Monitoring

Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.

Cluster-level Logging

A cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface

4.6 DEVELOP ON AZURE KUBERNETES SERVICE (AKS) WITH HELM

Helm is an open-source packaging tool that helps you install and manage the lifecycle of Kubernetes applications. Similar to Linux package managers like *APT* and *Yum*, Helm manages Kubernetes charts, which are packages of pre-configured Kubernetes resources.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, you can create a free account.
- Azure CLI installed.
- Helm v3 installed.
- Create an Azure Container Registry
- You'll need to store your container images in an Azure Container Registry (ACR) to run your application in your AKS cluster using Helm. Provide your own registry name unique within Azure and containing 5-50 alphanumeric characters. The *Basic* SKU is a cost-optimized entry point for development purposes that provides a balance of storage and throughput.
- The below example uses azacr create to create an ACR named *MyHelmACR* in *MyResourceGroup* with the *Basic* SKU.

  Azure CLI

az group create --name MyResourceGroup --location eastus

azacr create --resource-group MyResourceGroup --name MyHelmACR --sku Basic

Output will be similar to the following example. Take note of your *loginServer* value for your ACR since you'll use it in a later step. In the below example, *myhelmacr.azurecr.io* is the *loginServer* for *MyHelmACR*.

Console

{

"adminUserEnabled": false,

"creationDate": "2019-06-11T13:35:17.998425+00:00",

"id":
"/subscriptions/<ID>/resourceGroups/MyResourceGroup/providers/Microsoft.ContainerRegistry/registries/MyHelmACR",

"location": "eastus",

```
  "loginServer": "myhelmacr.azurecr.io",
  "name": "MyHelmACR",
  "networkRuleSet": null,
  "provisioningState": "Succeeded",
  "resourceGroup": "MyResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```


4.6.1 Create an AKS cluster

Your new AKS cluster needs access to your ACR to pull the container images and run them. Use the following command to:

- Create an AKS cluster called *MyAKS* and attach *MyHelmACR*.
- Grant the *MyAKS* cluster access to your *MyHelmACR* ACR.

Azure CLI

```
azaks create -g MyResourceGroup -n MyAKS --location eastus  --attach-acrMyHelmACR --generate-ssh-keys
```

4.6.2 Connect to your AKS cluster

To connect a Kubernetes cluster locally, use the Kubernetes command-line client, kubectl. kubectl is already installed if you use Azure Cloud Shell.

1. Install kubectl locally using the azaks install-cli command:
   azaks install-cli
2. Configure kubectl to connect to your Kubernetes cluster using the azaks get-credentials command. The following command example gets credentials for the AKS cluster named *MyAKS* in the *MyResourceGroup*:
   azaks get-credentials --resource-groupMyResourceGroup--nameMyAKS

4.7 BUILD AND PUSH THE SAMPLE APPLICATION TO THE ACR

Using the preceding Dockerfile, run the azacr build command to build and push an image to the registry. The . at the end of the command sets the location of the Dockerfile (in this case, the current directory).

```
azacr build --image azure-vote-front:v1 \
        --registry MyHelmACR \
        --file Dockerfile .
```

Create your Helm chart

Generate your Helm chart using the helm create command.

Console

helm create azure-vote-front

Update *azure-vote-front/Chart.yaml* to add a dependency for the *redis* chart from the https://charts.bitnami.com/bitnami chart repository and update appVersion to v1. For example:

Yml

apiVersion: v2

name: azure-vote-front

description: A Helm chart for Kubernetes

dependencies:

- name: redis

version: 14.7.1

repository: https://charts.bitnami.com/bitnami

...

 # This is the version number of the application being deployed. This version number should be

 # incremented each time you make changes to the application.

appVersion: v1

Update your helm chart dependencies using helm dependency update:

Console

helm dependency update azure-vote-front

Update *azure-vote-front/values.yaml*:

- Add a *redis* section to set the image details, container port, and deployment name.
- Add a *backendName* for connecting the frontend portion to the *redis* deployment.
- Change *image.repository* to <loginServer>/azure-vote-front.
- Change *image.tag* to v1.
- Change *service.type* to *LoadBalancer*.

For example:

yml

```
# Default values for azure-vote-front.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount:1
backendName:azure-vote-backend-master
redis:
  image:
    registry:mcr.microsoft.com
repository:oss/bitnami/redis
    tag:6.0.8
fullnameOverride:azure-vote-backend
  auth:
enabled:false

image:
repository:myhelmacr.azurecr.io/azure-vote-front
pullPolicy:IfNotPresent
  tag:"v1"
...
service:
type:LoadBalancer
  port:80
...
```

Add an env section to *azure-vote-front/templates/deployment.yaml* for passing the name of the *redis* deployment.

yml

```
...
     containers:
       - name:{{.Chart.Name}}
securityContext:
{{-toYaml.Values.securityContext| nindent 12 }}
         image:"{{ .Values.image.repository }}:{{ .Values.image.tag |
default .Chart.AppVersion }}"
imagePullPolicy:{{.Values.image.pullPolicy}}
         env:
         - name:REDIS
           value:{{.Values.backendName}}
...
```

Run your Helm chart

Install your application using your Helm chart using the helm install command.

Console
helm install azure-vote-front azure-vote-front/

It takes a few minutes for the service to return a public IP address. Monitor progress using the kubectl get service command with the --watch argument.

Console
$kubectl get service azure-vote-front --watch
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
azure-vote-front LoadBalancer 10.0.18.228 <pending>
80:32021/TCP 6s
...
azure-vote-front LoadBalancer 10.0.18.228 52.188.140.81
80:32021/TCP 2m6s
Navigate to your application's load balancer in a browser using
the <EXTERNAL-IP> to see the sample application.

4.8 MONITORING AZURE KUBERNETES SERVICE (AKS) WITH AZURE MONITOR

This scenario describes how to use Azure Monitor to monitor the health and performance of Azure Kubernetes Service (AKS). It includes collection of telemetry critical for monitoring, analysis and visualization of collected data to identify trends, and how to configure alerting to be proactively notified of critical issues.

Scope of the scenario

This scenario is intended for customers using Azure Monitor to monitor AKS. It does not include the following, although this content may be added in subsequent updates to the scenario.

- Monitoring of Kubernetes clusters outside of Azure except for referring to existing content for Azure Arc enabled Kubernetes.
- Monitoring of AKS with tools other than Azure Monitor except to fill gaps in Azure Monitor and Container Insights.

AKS generates platform metrics and resource logs, like any other Azure resource, that you can use to monitor its basic health and performance. Enable Container insights to expand on this monitoring. Container insights is a feature in Azure Monitor that monitors the health and performance of managed Kubernetes clusters hosted on AKS in addition to other cluster configurations. Container insights provides interactive views and workbooks that analyze collected data for a variety of monitoring scenarios.

Prometheus and Grafana are CNCF backed widely popular open source tools for kubernetes monitoring. AKS exposes many metrics in Prometheus format which makes Prometheus a popular choice for monitoring. Container insights has native integration with AKS, collecting

critical metrics and logs, alerting on identified issues, and providing visualization with workbooks. It also collects certain Prometheus metrics, and many native Azure Monitor insights are built-up on top of Prometheus metrics. Container insights complements and completes E2E monitoring of AKS including log collection which Prometheus as stand-alone tool doesn't provide. Many customers use Prometheus integration and Azure Monitor together for E2E monitoring.

4.8.1 Configure monitoring

The following sections describe the steps required to configure full monitoring of your AKS cluster using Azure Monitor.

Create Log Analytics workspace

You require at least one Log Analytics workspace to support Container insights and to collect and analyze other telemetry about your AKS cluster. There is no cost for the workspace, but you do incur ingestion and retention costs when you collect data.

If you're just getting started with Azure Monitor, then start with a single workspace and consider creating additional workspaces as your requirements evolve. Many environments will use a single workspace for all the Azure resources they monitor. You can even share a workspace used by Azure Security Center and Azure Sentinel, although many customers choose to segregate their availability and performance telemetry from security data.

Enable container insights

When you enable Container insights for your AKS cluster, it deploys a containerized version of the Log Analytics agent that sends data to Azure Monitor. There are multiple methods to enable it depending whether you're working with a new or existing AKS cluster.

Configure collection from Prometheus

Container insights allows you to collect certain Prometheus metrics in your Log Analytics workspace without requiring a Prometheus server. You can analyze this data using Azure Monitor features along with other data collected by Container insights.

Collect resource logs

The logs for AKS control plane components are implemented in Azure as resource logs. Container insights doesn't currently use these logs, so you do need to create your own log queries to view and analyze them. You need to create a diagnostic setting to collect resource logs. Create multiple diagnostic settings to send different sets of logs to different locations. There is a cost for sending resource logs to a workspace, so you should only collect those log categories that you intend to use. Send logs to an Azure storage account to reduce costs if you need to retain the information but don't require it to be readily available for analysis. If you're unsure about which resource logs to initially enable, use the

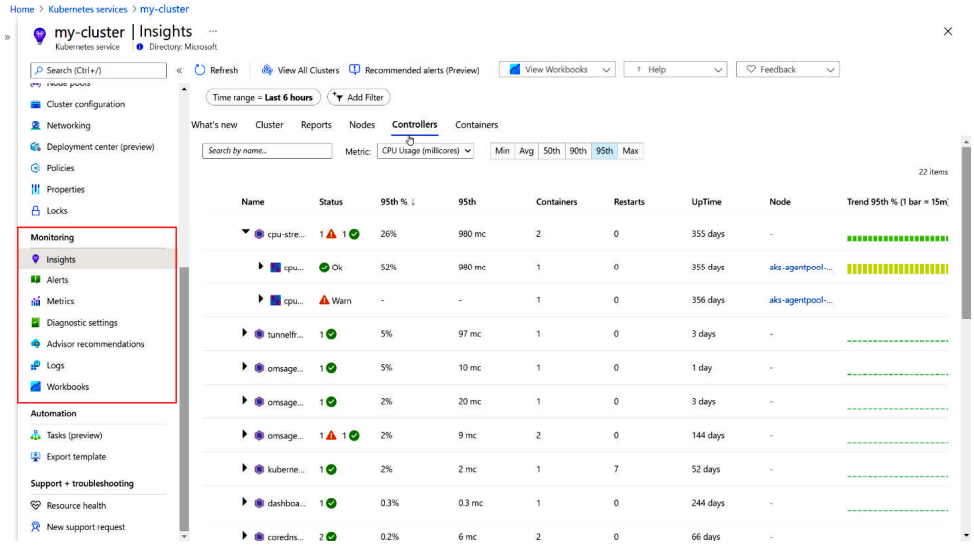
recommendations in the following table 4.1 which are based on the most common customer requirements. Enable the other categories if you later find that you require this information.

Table 4.1 Resource Logs

Category	Enable?	Destination
cluster-autoscaler	Enable if autoscale is enabled	Log Analytics workspace
guard	Enable if Azure Active Directory is enabled	Log Analytics workspace
kube-apiserver	Enable	Log Analytics workspace
kube-audit	Enable	Azure storage. This keeps costs to a minimum yet retains the audit logs if they're required by an auditor.
kube-audit-admin	Enable	Log Analytics workspace
kube-controller-manager	Enable	Log Analytics workspace
kube-scheduler	Disable	
AllMetrics	Enable	Log Analytics workspace

Access Azure Monitor features

Access Azure Monitor features for all AKS clusters in your subscription from the **Monitoring** menu in the Azure portal or for a single AKS cluster from the **Monitor** section of the **Kubernetes services** menu. The screenshot figure 4.1 below shows the cluster monitor menu.



Figurer 4.1 Cluster monitor Menu

Menu option	Description
Insights	Opens container insights for the current cluster. Select **Containers** from the **Monitor** menu to open container insights for all clusters.
Alerts	Views alerts for the current cluster.
Metrics	Open metrics explorer with the scope set to the current cluster.
Diagnostic settings	Create diagnostic settings for the cluster to collect resource logs.
Advisor	recommendations Recommendations for the current cluster from Azure Advisor.
Logs	Open Log Analytics with the scope set to the current cluster to analyze log data and access prebuilt queries.
Workbooks	Open workbook gallery for Kubernetes service.

Prometheus

Prometheus is free and an open-source event monitoring tool for containers or microservices. Prometheus collects numerical data based on time series. The Prometheus server works on the principle of scraping. This invokes the metric endpoint of the various nodes that have been configured to monitor. These metrics are collected in regular timestamps and stored locally. The endpoint that was used to discard is exposed on the node.

1.Prometheus Data Retention

Prometheus data retention time is 15 days by default. The lowest retention period is 2hour. If you retain the data for the highest period more disk space will be used as there will be more data. The lowest retention period can be used when configuring remote storage for Prometheus.

2. Prometheus with Grafana

Grafana is a multi-platform visualization software that provides us a graph, the chart for a web connected to the data source. Prometheus has it's own built-in browser expression but Grafana is the industry's most powerful visualization software. Grafana has out of the box integration with Prometheus.

Grafana

Grafana is a multi-platform visualization software available since 2014. Grafana provides us a graph, the chart for a web-connected to the data source. It can query or visualize your data source, it doesn't matter where they are stored.

1. Visualize

Swift and extensible client-side graphs with a number of options. There are many plugins for many different ways to visualize metrics and logs. You will use custom kubernetes metrics to plot them in the graph we will see that in the latter section

2. Explore Metrics

In this article, the Kube state metric list to visually see in the Grafana graph. Split view and compare different time ranges, queries, and data sources

3. Explore Logs

Experience the magic of switching from metrics to logs with preserved label filters. Quickly search through all your logs or stream them live.

SUMMARY

Kubernetes was created by Google based on lessons learned running containers at scale for many years. It was donated to the community as an open-source project and is now the industry standard API for deploying and managing cloud-native applications. It runs on any cloud or on-premises data center and abstracts the underlying infrastructure. This allows you to build hybrid clouds, as well as migrate easily between cloud platforms. It's open-sourced under the Apache 2.0 license and lives within the Cloud Native Computing Foundation (CNCF). Kubernetes is a fast-moving project under active development. But don't let this put you off – embrace it. Change is the new normal.

QUESTION BANK

1. What is Kubernetes?
2. What Kubernetes provides you with?
3. What is Microsoft Azure Kubernetes Services ?
4. What are the Azure Kubernetes Service Features?
5. What is the Advantages of AKS?
6. What are the Kubernetes Components?
7. How to Develop on Azure Kubernetes Service (AKS) with Helm

REFERENCE

1) https://github.com/indrabasak/Books/blob/master/Kubernetes%20in%20Action.pdf
2) https://github.com/Leverege/kubernetes-book/blob/master/An%20Introduction%20to%20Kubernetes%20%5BFeb%202019%5D.pdf
3) https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/pivotal/vmware-demystifying-kubernetes-overcoming-misconceptions-whitepaper.pdf
   https://www.kasten.io/kubernetes/resources/white-papers/5-kubernetes-backup-best-practices

❖❖❖❖

5

SECURING MICROSERVICE

Unit Structure :

5.0 OBJECTIVES

After going through this unit, you will be able to:

- Authentication in microservices
- Implenting security using API gateway pattern
- Creating application using Ocrlot and
- Securing APIs with Azure AD

5.1 INTRODUCTION

5.1.1 Monolithic Application Authentication and Authorization

It has been confusing to differentiate between authentication and authorization. In fact, it is very simple.

- Authentication: Refers to verify who you are, so you need to use username and password for authentication.
- Authorization: Refers to what you can do, for example access, edit or delete permissions to some documents, and this happens after verification passes.

In the monolithic architecture as show in figure 5.1, the entire application is a process. In the application, a security module is generally used to implement user authentication and authorization.

When the user logs in, the security module of the application authenticates the identity of the user. After verifying that the user is legitimate, a session is created for the user, and a unique session ID is

associated with the session. A session stores login user information such as User name, Role, and Permission. The server returns the Session Id to the client. The client records the Session Id as a cookie and sends it to the application in subsequent requests. The application can then use the Session Id to verify the user's identity, without having to enter a user name and password for authentication each time.

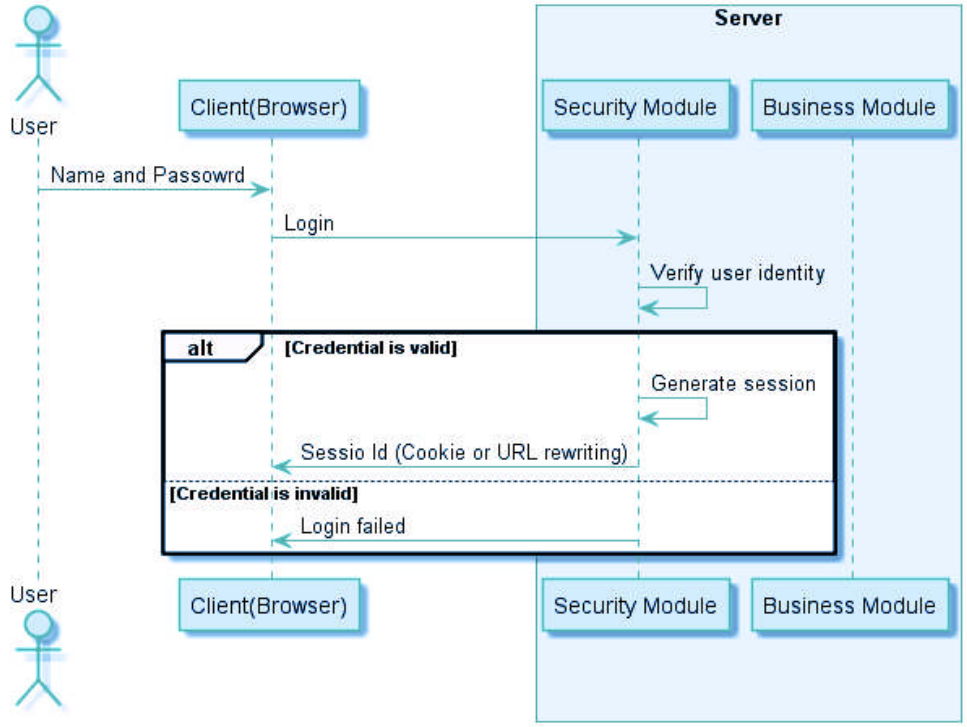


Figure 5.1Monolithic application user login authentication diagram

When the client accesses the application as shown in figure 5.2, Session Id is sent to the application along with the HTTP request. The security module generally processes all received client requests through an authorization interceptor. This interceptor first determines whether the Session Id exists. If the Session Id exists, it knows that the user has logged in. Then, by querying the user rights, it is determined whether the user can execute the request or not.

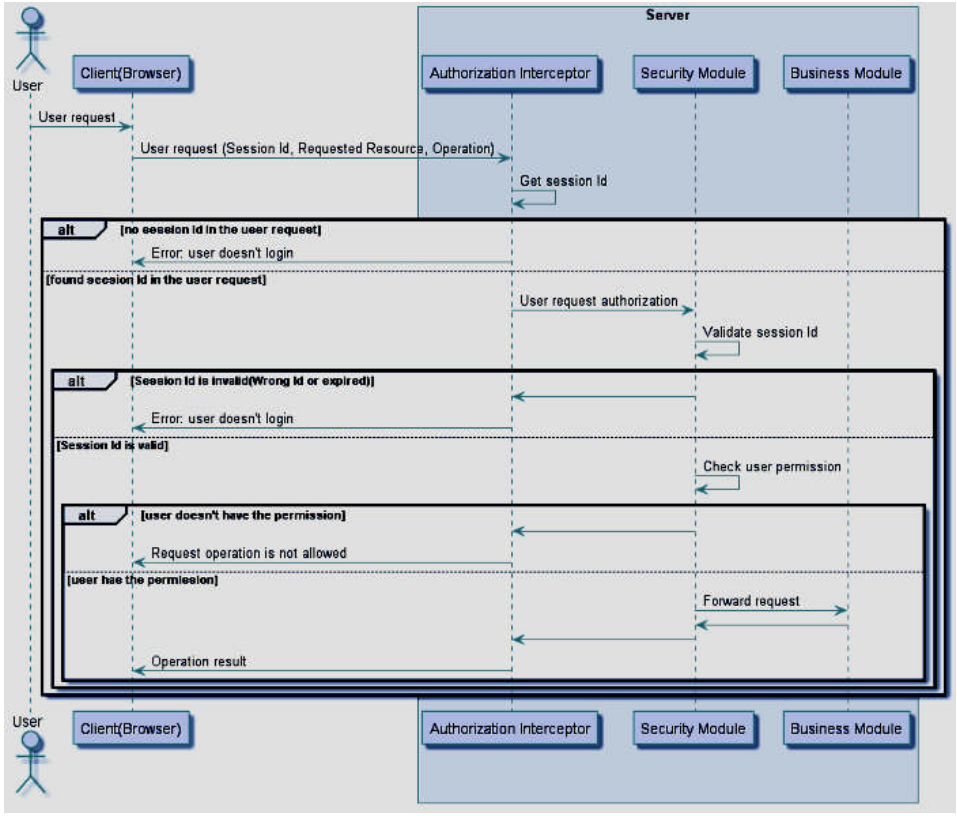


Figure 5.2 Monolithic application user request authorization diagram

5.2 MICROSERVICES AUTHENTICATION AND AUTHORIZATION PROBLEMS

Under the microservice architecture, an application is split into multiple microservice processes, and each microservice implements the business logic of one module in the original single application. After the application is split, the access request for each microservice needs to be authenticated and authorized. If you reference to the implementation of Monolithic application, you will encounter the following problems:

- Authentication and authorization logic needs to be handled in each microservice, and this part of the global logic needs to be implemented repeatedly in each microservice. Although we can use the code base to reuse part of the code, this will in turn cause all micro services to have a dependency on a particular code base and its version, affecting the flexibility of the microservice language/framework selection.
- Microservices should follow the principle of single responsibility. A microservice only handles a single business logic. The global logic of authentication and authorization should not be placed in the microservice implementation.
- HTTP is a stateless protocol. For the server, each time the user's HTTP request is independent. Stateless means that the server can send client requests to any node in the cluster as needed. The stateless design of HTTP has obvious benefits for load balancing. Because there is no state, user requests can be distributed to any server. For services

that do not require authentication, such as browsing news pages, there is no problem. However, many services, such as online shopping and enterprise management systems, need to authenticate the user's identity. Therefore, it is necessary to save the user's login status in a manner based on the HTTP protocol so as to prevent the user from needing to perform verification for each request. The traditional way is to use a session on the server side to save the user state. Because the server is stateful, it affects the horizontal expansion of the server.

- The authentication and authorization in the microservices architecture involves scenarios that are more complex, involving users accessing microservice applications, third-party applications accessing microservice applications, and multiple microservice applications accessing each other, and in each scenario, The following authentication and authorization schemes need to be considered to ensure the security of the application.

5.3MICROSERVICES AUTHENTICATION AND AUTHORIZATION TECHNICAL SOLUTIONS

1) Distributed Session Management

In order to make full use of benefits of the microservice architecture and to achieve the scalability and resiliency of the microservices, the microservices are preferably to be stateless.

This solution can be applied through different ways like:

Sticky Session

Which ensures that all requests from a specific user will be sent to the same server who handled the first request corresponding to that user, thus ensuring that session data is always correct for a certain user. However, this solution depends on the load balancer, and it can only meet the horizontally expanded cluster scenario, but when the load balancer is forced suddenly for any reason to shift users to a different server, all of the user's session data will be lost.

Session Replication

Means that each instance saves all session data, and synchronizes through the network. Synchronizing session data causes network bandwidth overhead. As long as the session data changes, the data needs to be synchronized to all other machines. The more instances, the more network bandwidth the synchronization brings.

Centralized Session Storage

Means that when a user accesses a microservice, user data can be obtained from shared session storage, ensuring that all microservices can read the same session data. In some scenarios, this scheme is very good, and the user login status is opaque. It is also a highly available and scalable solution. But the disadvantage of this solution is that shared

session storage requires a certain protection mechanism and therefore needs to be accessed through a secure way.

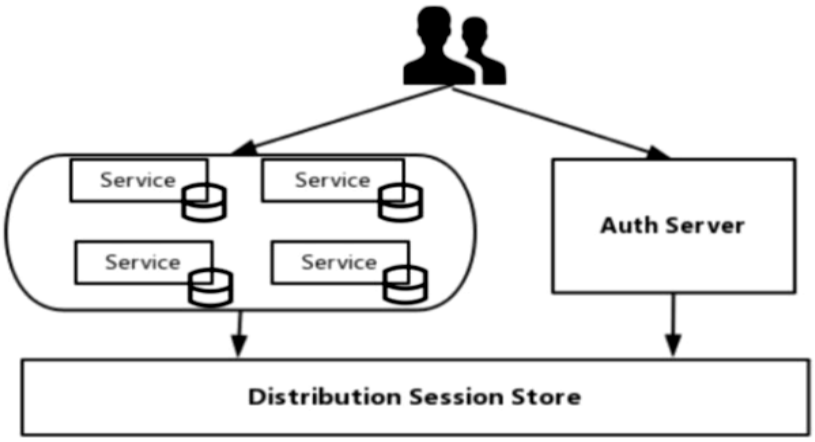


Figure 5.3 Distrubted Session Management

2. Client Token

The traditional way is to use a session on the server side to save the user state. Because the server is stateful, it has an impact on the horizontal expansion of the server. It is recommended to use Token to record user login status in the microservice architecture.

The main difference between Token and Session is where the storage is different. Sessions are stored centrally in the server; Tokens are held by the user themselves and are typically stored in the browser in the form of cookies. The Token holds the user's identity information, and each time the request is sent to the server, the server can therefore determine the identity of the visitor and determine whether it has access to the requested resource.

The Token is used to indicate the user's identity. Therefore, the content of the Token needs to be encrypted to avoid falsification by the requester or the third party. JWT (Json Web Token) is an open standard (RFC 7519) that defines the Token format, defines the Token content, encrypts it, and provides lib for various languages.

The structure of JWT Token is very simple and consists of three parts:

- **Header**
  header contains type, fixed value JWT. Then the Hash algorithm used by JWT.

{
"typ": "JWT",
"alg": "HS256"
}

- **Payload**
  includes standard information such as the user id, expiration date, and user name. It can also add user roles and user-defined information.

{
"id": 123,

"name": "Mena Meseha",
"is_admin": true,
"expire": 1558213420
}

- **Signature**
  Token's signature is used by the client to verify the Token's identity and also to verify the message wasn't changed along the way.

HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret
)

These three parts are combined using Base64 encoding and become Token strings that are eventually returned to the client, separated by ".", The token formed by the above example will be like this:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MTIzLCJuYW1lIjoiTWVuYSBNZXNlaGEiLCJpc19hZG1pbiI6dHJ1ZSwiZXhwaXJlIjoxNTU4MjEzNDIwfQ.Kmy_2WCPbpg-aKQmiLaKFLxb5d3rOC71DHexncH_AcQ

By using token for user authentication, the server does not save the user status. The client needs to send the token to the server for authentication every time the client requests it.

The basic flow of user authentication in token mode is as the following figure 5.4:

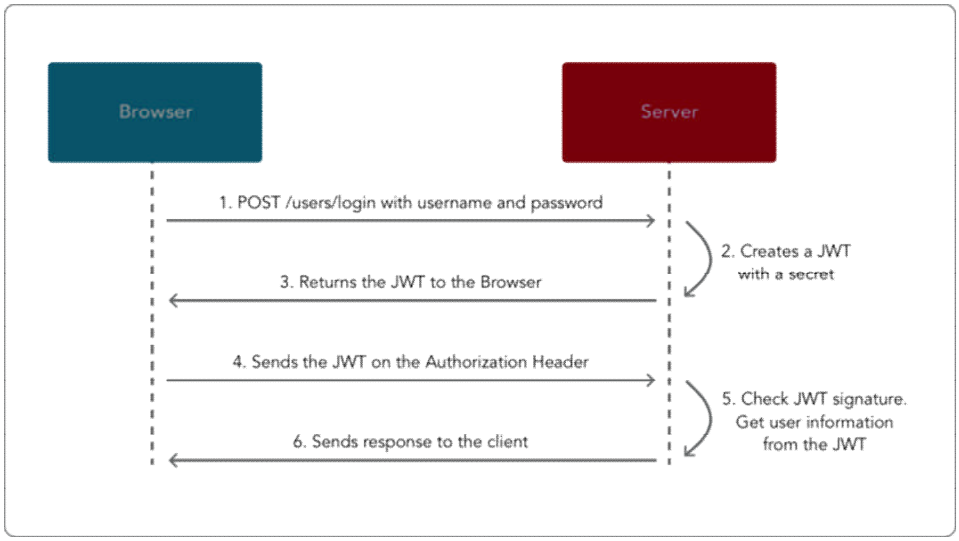


Figure 5.4 Client Token

3. Single sign-on

The idea of single sign-on is simple, that is, users only need to log in to the application once,and then they can access all the microservices in the application. This solution means that each user-oriented service must interact with the authentication service like the following figure 5.4:

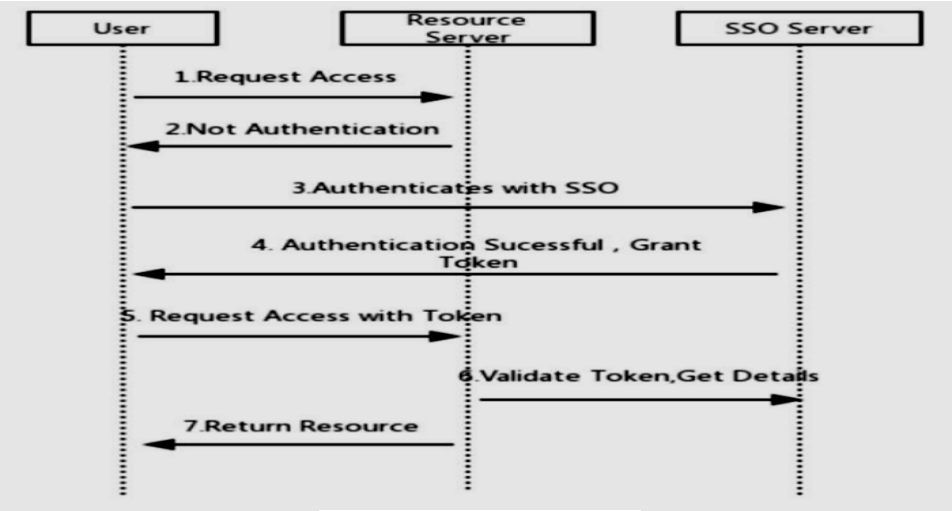


Figure 5.5 Single sign-on

This can result in a lot of very trivial network traffic, repeated work, and it may cause single point of failure. When there are dozens of micro-applications, the drawbacks of this solution will become more apparent.

4. Client Token with API Gateway

The authentication process of the user is similar to the basic process of token authentication. The difference is that the API Gateway is added as the entrance of the external request. This scenario means that all requests go through the API gateway, effectively hiding the microservices. On request, the API gateway translates the original user token into an opaque token that only itself can resolve like the following figure 5.6:

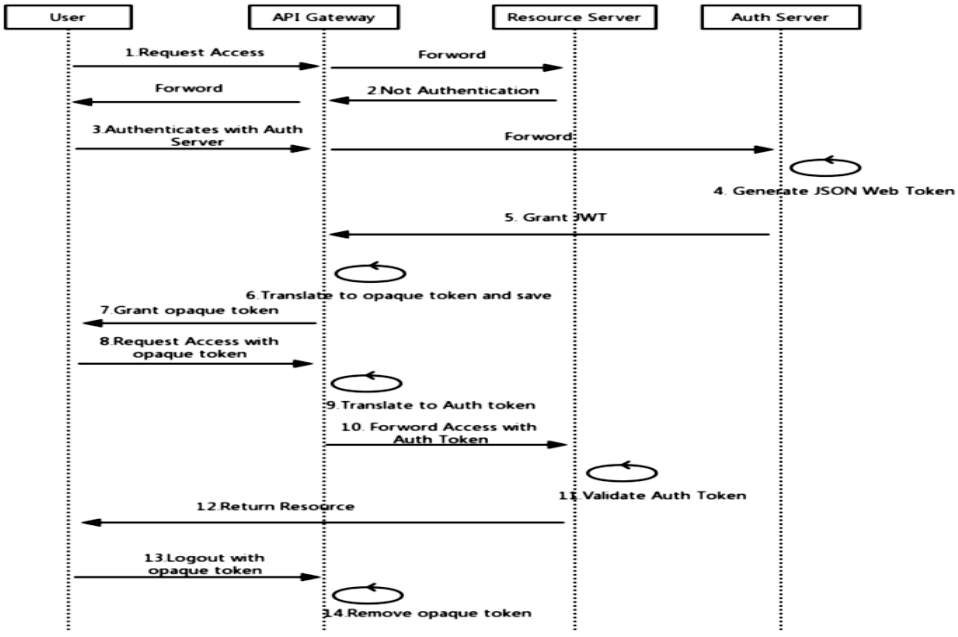


Figure 5.6Client Token with API Gateway solution

In this case, logging off is not a problem because the API gateway can revoke the user's token when it logs out and also it adds an extra protection to Auth Token from being decrypted by hiding it from the client.

5. Third-party application access

1. API Token

The third party uses an application-issued API Token to access the application's data. The Token is generated by the user in the application and provided for use by third-party applications. In this case, generally only third-party applications are allowed to access the user's own data of the Token, but not other users' sensitive private data.

For example, Github provides the Personal API Token function. Users can create a Token in Github's developer settings interface, and then use the Token to access the Github API. When creating a Token, you can set which data the Token can access to the user, such as viewing Repo information, deleting Repo, viewing user information, updating user information, and so on.

Using the API Token to Access the Github API is like the following command:

curl-u menameseha:f3kdfvf8e882424ed0f3bavmvdl88c01acd34eec https://api.github.com/user

The advantage of using the API Token instead of using the username/password directly to access the API is to reduce the risk of exposing the user's password, and to reclaim the token's permissions at any time without having to change the password.

2. OAuth

Some third-party applications need to access data from different users, or integrate data from multiple users. You may consider using OAuth. With OAuth, when a third-party application accesses a service, the application prompts the user to authorize a third-party application to use the corresponding access authority and generates a token for access according to the user's permissions.

In Github, for example, some third-party applications such as GitBook or Travis CI, are integrated via OAuth and Github. OAuth has different authentication processes for different scenarios. A typical authentication process is shown in the following figure5.7:

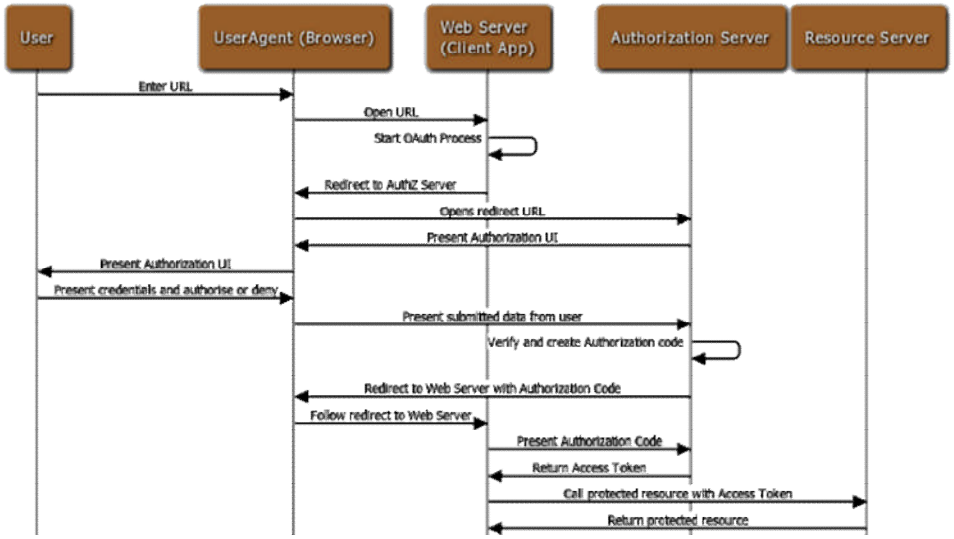


Figure 5.7OAuth authentication process

Someone may wonder why an Authorization Code is used to request Access Token, rather than returning the Access Token to the client directly from the authorization server. The reason why OAuth is designed in this way is to pass through the user agent (browser) during the process of redirecting to the client's Callback URL. If the Access Token is passed directly, there is a risk of being stolen.

By using the authorization code, the client directly interacts with the authorization server when applying for the access token, and the authorization server also authorize the client when processing the client's token request, so it's prevent others from forging the client's identity to use the authentication code.

When implementing user authentication of the microservice itself, OAuth may also be used to delegate user authentication of the microservice to a third-party authentication service provider.

The purpose of using OAuth for user authorization of third-party application access and microservices is different. The former is to authorize private data access rights of users in microservices to third-party applications. Microservices are authorization and resource servers in the OAuth architecture. The purpose of the latter is to integrate and utilize the OAuth authentication service provided by a well-known authentication provider, which simplifies the cumbersome registration operation, in this case the microservice act the role of the client in the OAuthArchitecture. Therefore, we need to distinguish between these two different scenarios so as to avoid misunderstandings.

6. Mutual Authentication

In addition to vertical traffic from users and third parties, there is a large amount of horizontal traffic between microservices. These traffic may be in the same local area network or across different data centers. Traffic

between these microservices exists by third parties. The danger of sniffing and attacking also requires security controls.

Through mutual SSL, mutual authentication between microservices can be achieved, and data transmission between microservices can be encrypted through TLS. A certificate needs to be generated for each microservice, and the microservices are authenticated with each other's certificates. In the microservice operating environment, there may be a large number of microservice instances, and the microservice instances often change dynamically, such as adding service instances as the level expands. In this case, creating and distributing certificates for each service becomes very difficult. We can create a private certificate center (Internal PKI/CA) to provide certificate management for various microservices such as issuing, revoking, and updating.

5.4 IMPLEMENTING YOUR API GATEWAYS WITH OCELOT

Ocelot is a .NET API Gateway. This project is aimed at people using .NET running a microservices / service-oriented architecture that need a unified point of entry into their system. However, it will work with anything that speaks HTTP and run on any platform that ASP.NET Core supports.

Ocelot is a bunch of middleware in a specific order.

Ocelot manipulates the HttpRequest object into a state specified by its configuration until it reaches a request builder middleware where it creates a HttpRequestMessage object which is used to make a request to a downstream service. The middleware that makes the request is the last thing in the Ocelot pipeline. It does not call the next middleware. The response from the downstream service is retrieved as the requests goes back up the Ocelot pipeline. There is a piece of middleware that maps the HttpResponseMessage onto the HttpResponse object and that is returned to the client. That is basically it with a bunch of other features!

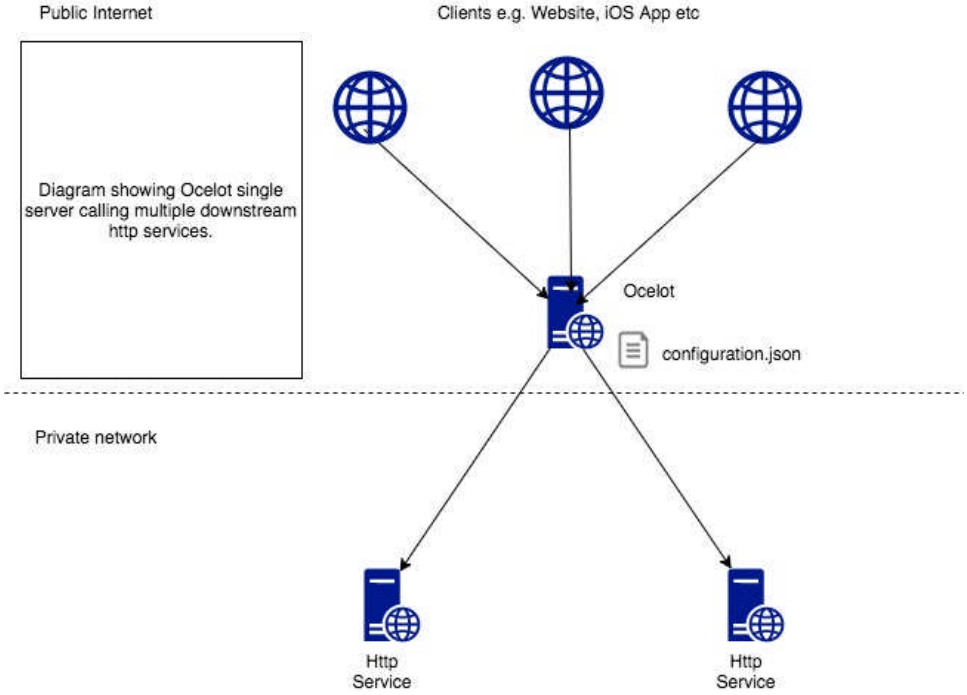
Features

- Routing
- Request Aggregation
- Service Discovery with Consul & Eureka
- Service Fabric
- Kubernetes
- WebSockets
- Authentication
- Authorization
- Rate Limiting
- Caching
- Retry policies / QoS
- Load Balancing
- Logging / Tracing / Correlation

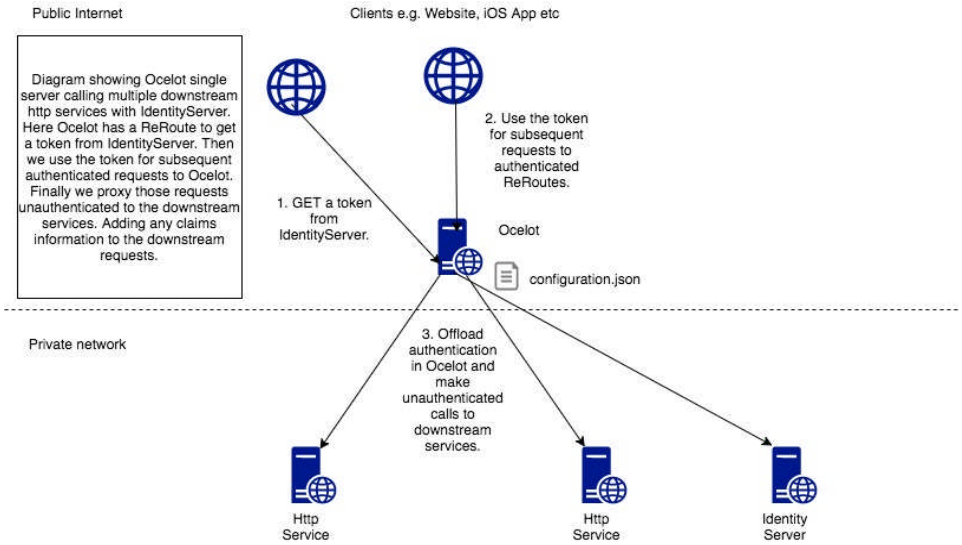
- Headers / Method / Query String / Claims Transformation
- Custom Middleware / Delegating Handlers
- Configuration / Administration REST API
- Platform / Cloud Agnostic

The following are configurations that you use when deploying Ocelot.

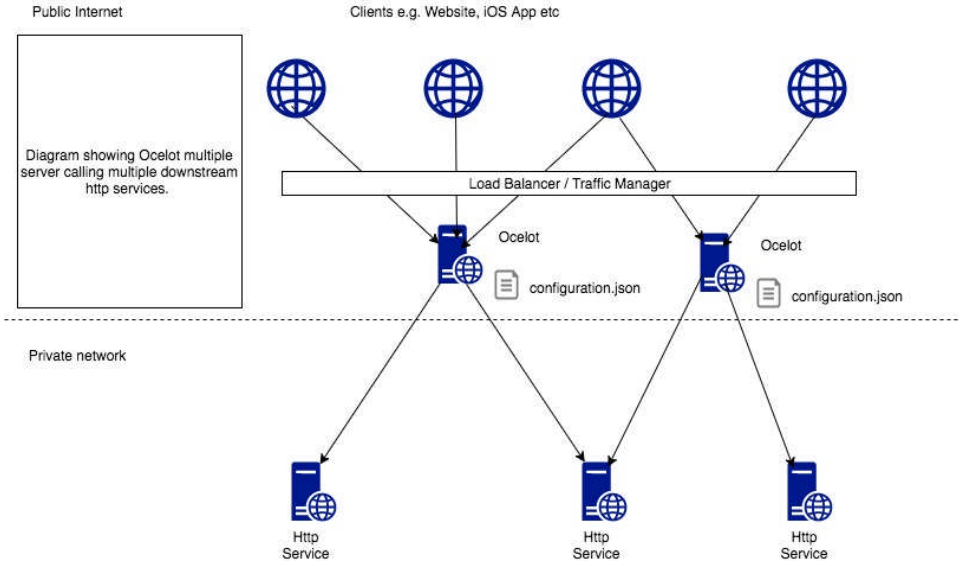
Basic Implementation



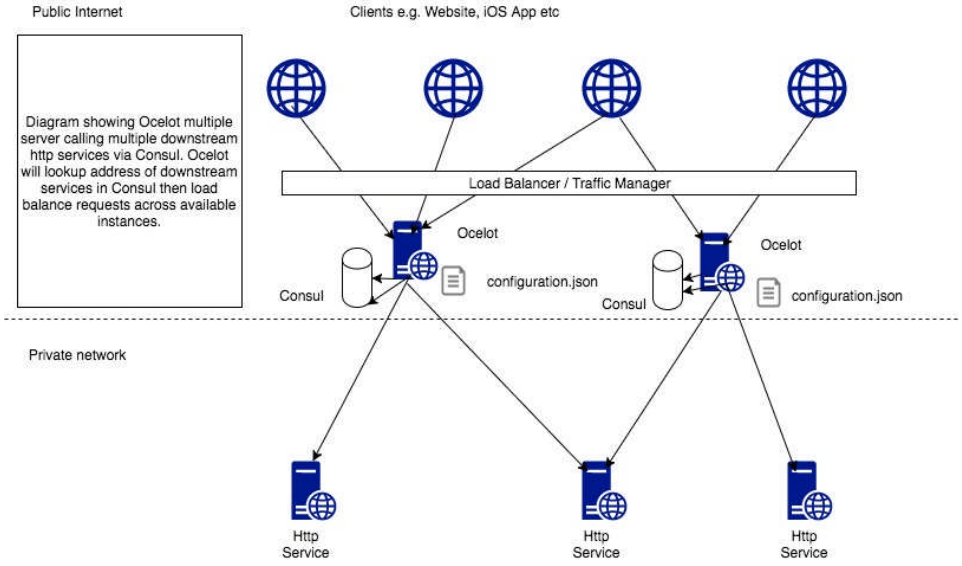
With Identity Server



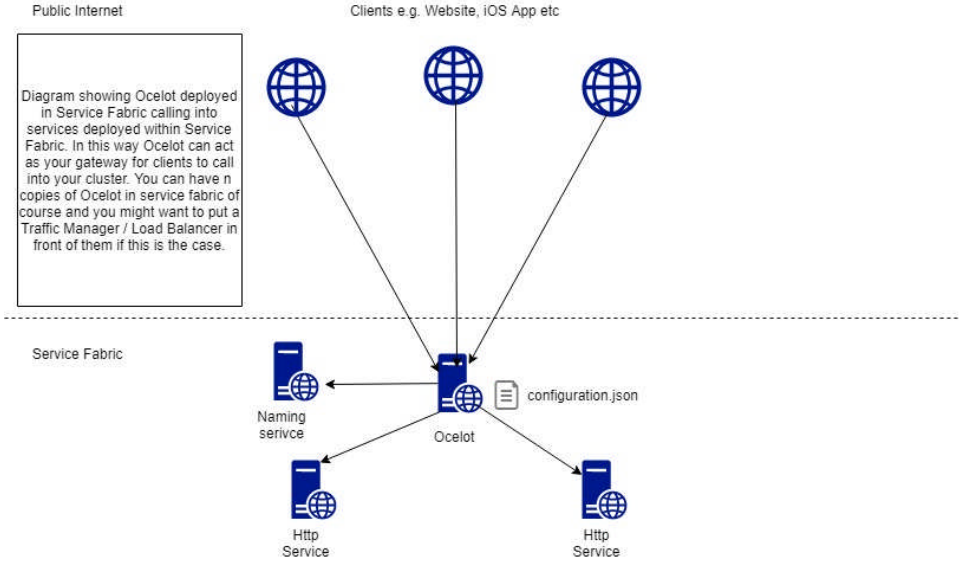
Multiple Instances



With Consul



With Service Fabric



5.5 SECURING APIS WITH AZURE AD.

Microsoft Azure Active Directory (AD) is PaaS service available to every Azure subscription, this service is used to store information about users and organizational structure. We'll use this service as our **Authority** service which will be responsible to secure our **Resource** (Web API) and issue access tokens and refresh tokens using OAuth 2 Code flow grant. The resource (Web API) should be consumed by a **Client**, so the client will be requesting the data from the resource (Web API), but in order for this request to be accepted by the resource, the client must send a valid access token obtained from the Authority service (Azure AD) with each request.

5.5.1 Register an application in Azure AD to represent the API

To protect an API with Azure AD, first register an application in Azure AD that represents the API. The following steps use the Azure portal to register the application.

Go to the Azure portal to register your application. Search for and select **App registrations**.

1. Select **New registration**.
2. When the **Register an application page** appears, enter your application's registration information:
   - In the **Name** section, enter a meaningful application name that will be displayed to users of the app, such as *backend-app*.
   - In the **Supported account types** section, select an option that suits your scenario.
3. Leave the **Redirect URI** section empty.
4. Select **Register** to create the application.
5. On the app **Overview** page, find the **Application (client) ID** value and record it for later.
6. Select **Expose an API** and set the **Application ID URI** with the default value. Record this value for later.
7. Select the **Add a scope** button to display the **Add a scope** page. Then create a new scope that's supported by the API (for example, Files. Read).
8. Select the **Add scope** button to create the scope. Repeat this step to add all scopes supported by your API.
9. When the scopes are created, make a note of them for use in a subsequent step.

5.5.2 Register another application in Azure AD to represent a client application

Every client application that calls the API needs to be registered as an application in Azure AD. In this example, the client application is the **Developer Console** in the API Management developer portal.

To register another application in Azure AD to represent the Developer Console:

1. Go to the Azure portal to register your application.
2. Search for and select **App registrations**.
3. Select **New registration**.
4. When the **Register an application page** appears, enter your application's registration information:
   - In the **Name** section, enter a meaningful application name that will be displayed to users of the app, such as *client-app*.
   - In the **Supported account types** section, select **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**.
5. In the **Redirect URI** section, select Web and leave the URL field empty for now.
6. Select **Register** to create the application.
7. On the app **Overview** page, find the **Application (client) ID** value and record it for later.
8. Create a client secret for this application to use in a subsequent step.
   - From the list of pages for your client app, select **Certificates & secrets**, and select **New client secret**.
   - Under **Add a client secret**, provide a **Description**. Choose when the key should expire, and select **Add**.

When the secret is created, note the key value for use in a subsequent step.

5.5.3 Grant permissions in Azure AD

Now that you have registered two applications to represent the API and the Developer Console, grant permissions to allow the client-app to call the backend-app.

1. Go to the Azure portal to grant permissions to your client application. Search for and select **App registrations**.
2. Choose your client app. Then in the list of pages for the app, select **API permissions**.
3. Select **Add a Permission**.
4. Under **Select an API**, select **My APIs**, and then find and select your backend-app.
5. Under **Delegated Permissions**, select the appropriate permissions to your backend-app, then select **Add permissions**.
6. Optionally, on the **API permissions** page, select **Grant admin consent for <your-tenant-name>** to grant consent on behalf of all users in this directory.

5.5.4 Enable OAuth 2.0 user authorization in the Developer Console

At this point, you have created your applications in Azure AD, and have granted proper permissions to allow the client-app to call the backend-app.

In this example, the Developer Console is the client-app. The following steps describe how to enable OAuth 2.0 user authorization in the Developer Console.

1. In Azure portal, browse to your API Management instance.
2. Select **OAuth 2.0** > **Add**.
3. Provide a **Display name** and **Description**.
4. For the **Client registration page URL**, enter a placeholder value, such as http://localhost. The **Client registration page URL** points to a page that users can use to create and configure their own accounts for OAuth 2.0 providers that support this. In this example, users do not create and configure their own accounts, so you use a placeholder instead.
5. For **Authorization grant types**, select **Authorization code**.
6. Specify the **Authorization endpoint URL** and **Token endpoint URL**. Retrieve these values from the **Endpoints** page in your Azure AD tenant. Browse to the **App registrations** page again, and select **Endpoints**.
7. Copy the **OAuth 2.0 Authorization Endpoint**, and paste it into the **Authorization endpoint URL** text box. Select **POST** under Authorization request method.
8. Copy the **OAuth 2.0 Token Endpoint**, and paste it into the **Token endpoint URL** text box.

   Important Note

   Use either **v1** or **v2** endpoints. However, depending on which version you choose, the below step will be different. We recommend using v2 endpoints.
9. If you use **v1** endpoints, add a body parameter named **resource**. For the value of this parameter, use **Application ID** of the back-end app.
10. If you use **v2** endpoints, use the scope you created for the backend-app in the **Default scope** field. Also, make sure to set the value for the accessTokenAcceptedVersion property to 2 in your application manifest.
11. Next, specify the client credentials. These are the credentials for the client-app.
12. For **Client ID**, use the **Application ID** of the client-app.
13. For **Client secret**, use the key you created for the client-app earlier.
14. Immediately following the client secret is the **redirect_url** for the authorization code grant type. Make a note of this URL.
15. Select **Create**.
16. Go back to your client-app registration in Azure Active Directory and select **Authentication**.

17. Under **Platform configurations** click on **Add a platform**, and select the type as **Web**, paste the **redirect_url** under **Redirect URI**, and then click on **Configure** button to save.

Now that you have configured an OAuth 2.0 authorization server, the Developer Console can obtain access tokens from Azure AD.

The next step is to enable OAuth 2.0 user authorization for your API. This enables the Developer Console to know that it needs to obtain an access token on behalf of the user, before making calls to your API.

1. Browse to your API Management instance, and go to **APIs**.
2. Select the API you want to protect. For example, Echo API.
3. Go to **Settings**.
4. Under **Security**, choose **OAuth 2.0**, and select the OAuth 2.0 server you configured earlier.
5. Select **Save**.

Successfully call the API from the developer portal

Note

This section does not apply to the **Consumption** tier, which does not support the developer portal.

Now that the OAuth 2.0 user authorization is enabled on your API, the Developer Console will obtain an access token on behalf of the user, before calling the API.

1. Browse to any operation under the API in the developer portal, and select **Try it**. This brings you to the Developer Console.
2. Note a new item in the **Authorization** section, corresponding to the authorization server you just added.
3. Select **Authorization code** from the authorization drop-down list, and you are prompted to sign in to the Azure AD tenant. If you are already signed in with the account, you might not be prompted.
4. After successful sign-in, an Authorization header is added to the request, with an access token from Azure AD. The following is a sample token (Base64 encoded):

Authorization:

Bearer

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlNTUWRoSTFjS3ZoUUVE
U0p4RTJnR1lzNDBRMCIsImtpZCI6IlNTUWRoSTFjS3ZoUUVEU0p4RTJnR1lzN
DBRMCJ9.eyJhdWQiOiIxYzg2ZWVmNC1jMjZkLTRiNGUtODEzNy0wYjBiZTE
yM2NhMGMiLCJpc3MiOiJodHRwczovL3N0cy53aW5kb3dzLm5ldC80NDc4ODk
yMC05Yjk3LTRmOGItODIwYS0yMTFiMTMzZDk1MzgvIiwiaWF0IjoxNTIxMT
UyNjMzLCJuYmYiOjE1MjExNTI2MzMsImV4cCI6MTUyMTE1NjUzMywiYWN
yIjoiMSIsImFpbyI6IkFWUUFxLzhHQUFBQUptVzkzTFd6dVArcGF4ZzJPeGE1c
Gp2V1NXV1ZSVnd1ZXZ5QU5yMlNkc0tkQmFWNnNjcHZsbUpmT1dDOThscUJ
JMDhXdlB6cDdlenpJdzJLai9MdWdXWWdydHhkM1lmaDlYSGpXeFVaWk9JPSI
sImFtciI6WyJyc2EiXSwiYXBwaWQiOiJhYTY5ODM1OC0yMWEzLTRhYTQtYjI
3OC1mMzI2NTMzMDUzZTkiLCJhcHBpZGFjciI6IjEiLCJlbWFpbCI6Im1pamlhb
mdAbWljcm9zb2Z0LmNvbSIsImZhbWlseV9uYW1lIjoiSmlhbmciLCJnaXZlbl9uY
W1lIjoiTWlhbyIsImlkcCI6Imh0dHBzOi8vc3RzLndpbmRvd3MubmV0LzcyZjk4O

GJmLTg2ZjEtNDFhZi05MWFiLTJkN2NkMDExZGI0Ny8iLCJpcGFkZHIiOiIxMzEuMTA3LjE3NC4xNDAiLCJuYW1lIjoiTWlhbyBKaWFuZyIsIm9pZCI6IjhiMTU4ZDEwLWVmZGItNDUxMS1iOTQzLTczOWZkYjMxNzAyZSIsInNjcCI6InVzZXJfaW1wZXJzb25hdGlvbiIsInN1YiI6IkFGaWtvWFk1TEV1LTNkbk1pa3Z3MUJzQUx4SGIybV9IaVJjaHVfSEM1aGciLCJ0aWQiOiI0NDc4ODkyMC05Yjk3LTRmOGItODIwYS0yMTFiMTMzZDk1MzgiLCJ1bmlxdWVfbmFtZSI6Im1pamlhbmdAbWljcm9zb2Z0LmNvbSIsInV0aSI6ImFQaTJxOVZ6ODBXdHNsYjRBMzBCQUEiLCJ2ZXIiOiIxLjAifQ.agGfaegYRnGj6DM_-N_eYulnQdXHhrsus45QDuApirETDR2P2aMRxRioOCR2YVwn8pmpQ1LoAhddcYMWisrw_qhaQr0AYsDPWRtJ6x0hDk5teUgbix3gazb7F-TVcC1gXpc9y7j77Ujxcq9z0r5lF65Y9bpNSefn9Te6GZYG7BgKEixqC4W6LqjtcjuOuW-ouy6LSSox71Fj4Ni3zkGfxX1T_jiOvQTd6BBltSrShDm0bTMefoyX8oqfMEA2ziKjwvBFrOjO0uK4rJLgLYH4qvkR0bdF9etdstqKMo5gecarWHNzWi_tghQu9aE3Z3EZdYNI_ZGM-Bbe3pkCfvEOyA

5. Select **Send** to call the API successfully.

5.6 CONFIGURE A JWT VALIDATION POLICY TO PRE-AUTHORIZE REQUESTS

At this point, when a user tries to make a call from the Developer Console, the user is prompted to sign in. The Developer Console obtains an access token on behalf of the user, and includes the token in the request made to the API.

However, what if someone calls your API without a token or with an invalid token? For example, try to call the API without the Authorization header, the call will still go through. The reason is that API Management does not validate the access token at this point. It simply passes the Authorization header to the back-end API.

Use the Validate JWT policy to pre-authorize requests in API Management, by validating the access tokens of each incoming request. If a request does not have a valid token, API Management blocks it. For example, add the following policy to the <inbound> policy section of the Echo API. It checks the audience claim in an access token, and returns an error message if the token is not valid.

xml

```
<validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-validation-error-message="Unauthorized. Access token is missing or invalid.">
<openid-config url="https://login.microsoftonline.com/{aad-tenant}/.well-known/openid-configuration" />
<required-claims>
<claim name="aud">
<value>{Application ID of backend-app}</value>
</claim>
</required-claims>
</validate-jwt>
```

SUMMARY

At its core, Azure replaces or supplements your on-premise infrastructure. However, it delivers a vast range of other services that improve the functioning of several departments in your organization and help you resolve critical business problems. For instance, you can get big data insights with Azure analytics, and manage your billions of IoT devices on a unified Azure platform, you can interact with your users with AI bots through various platforms, and get a secure and scalable cloud data storage. You can also automate your development, testing, and deployment with DevOps, and deliver content across the globe without facing any latency issues. These services are only a glimpse of what Azure can do for your business. Many enterprises across the globe are utilizing the capabilities of Microsoft Azure applications to optimize their business models as it revolutionizes overall infrastructure and application performance.

QUESTION BANK

1) Explain the Importance of the role and how many types of roles are available in Windows Azure?
2) Which service in Azure is used to manage resources in Azure?
3) What are virtual machine scale sets in Azure?
4) Why is Azure Active Directory used?
5) Name and explain some important applications of Microsoft Azure
6) How to Implementing your API Gateways with Ocelot
7) How to Securing APIs with Azure AD

REFERENCE

1) https://download.microsoft.com/DOWNLOAD/2/C/A/2CA4DC8E-021C-4D56-8529-DF4F71FF4A1B/9780735697225.PDF
2) https://ptgmedia.pearsoncmg.com/images/9780735697225/samplepages/9780735697225.pdf
3) https://docs.microsoft.com/en-in/azure/guides/developer/azure-developer-guide

❖❖❖❖

6

DATABASE DESIGN FOR MICROSERVICES & BUILDING MICROSERVICES ON AZURE STACK

Unit Structure :

6.0 OBJECTIVES

After going through this unit, you will be able to:

1) Understand difference between monolithic approach and Microservices approach
2) Understand different database options on MS Azure
3) Understand Azure stack, Offering IaaS,
4) Understand PaaS on-premises simplified and SaaS on Azure

6.1 INTRODUCTION

6.1.1 Monolithic Architecture

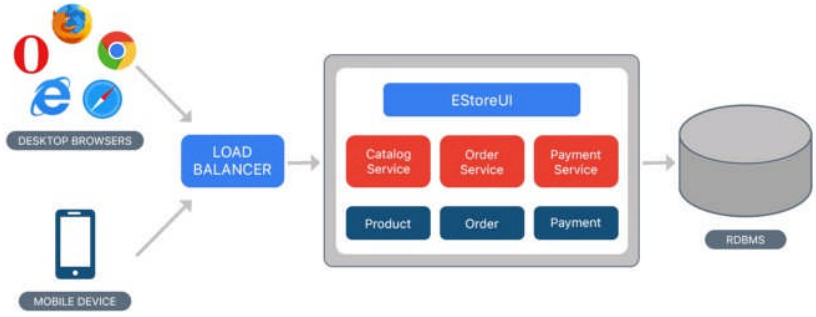
Monolith means composed all in one piece. The Monolithic application describes a single-tiered **software** application in which different components combined into a single program from a single platform. Components can be:

- Authorization — responsible for authorizing a user
- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).

- Business logic — the application's business logic.
- Database layer — data access objects responsible for accessing the database.
- Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
- Notification module — responsible for sending email notifications whenever needed.

6.1.2 Example for Monolithic Approach

Consider an example of Ecommerce application, that authorizes customer, takes an order, check products inventory, authorize payment and ships ordered products. This application consists of several components including e-Store User interface for customers (Store web view) along with some backend services to check products inventory, authorize and charge payments and shipping orders.



Despite having different components/modules/services, the application is built and deployed as one Application for all platforms (i.e. desktop, mobile and tablet) using RDBMS as a data source.

6.1.3 Benefits and Drawbacks of Monolithic Architecture.

Benefits:

- Simple to develop — At the beginning of a project it is much easier to go with Monolithic Architecture.
- Simple to test. For example, you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.
- Simple to deploy. You have to copy the packaged application to a server.
- Simple to scale horizontally by running multiple copies behind a load balancer.

Drawbacks:

- Maintenance — If Application is too large and complex to understand entirely, it is challenging to make changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Monolithic applications can also be challenging to scale when different modules have conflicting resource requirements.
- Reliability — Bug in any module (e.g. memory leak) can potentially bring down the entire process. Moreover, since all

instances of the application are identical, that bug impact the availability of the entire application

- Regardless of how easy the initial stages may seem, Monolithic applications have difficulty to adopting new and advance technologies. Since changes in languages or frameworks affect an entire application, it requires efforts to thoroughly work with the app details, hence it is costly considering both time and efforts.

6.1.4Microservices Architecture

Micro services are an approach to application development in which a large application is built as a suite of modular services (i.e. loosely coupled modules/components). Each module supports a specific business goal and uses a simple, well-defined interface to communicate with other sets of services.

Instead of sharing a single database as in Monolithic application, each micro service has its own database. Having a database per service is essential if you want to benefit from micro services, because it ensures **loose coupling**. Each of the services has its own database. Moreover, a service can use a type of database that is best suited to its needs.

Consider the same example of the e-commerce application, which consists of several components/modules. Define each component/module as a separate **loosely coupled** service depending on the requirement, which may collaborate with each other based on the scenario. We can have following services for a complete application:

- Authorization Service — Responsible for authorizing customer.
- Order Service — takes an order and process it.
- Catalog Service — Manage products and check products inventory.
- Cart Service — Manage user cart, this service can utilize Catalog service as a data source.
- Payment Service — Manage and Authorize payments.
- Shipping Service — Ships ordered products.

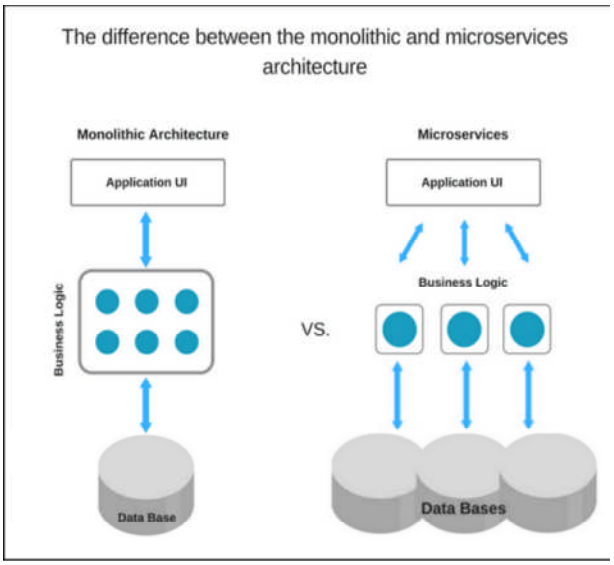
6.1.5 Benefits and Drawbacks of Monolithic Architecture.

Benefits:

- Microservices Enables the continuous delivery and deployment of large, complex applications.
- Better testability — services are smaller and faster to test.
- Better deployability — services can be deployed independently.
- It enables you to organize the development effort around multiple teams. Each team is responsible for one or more single service. Each team can develop, deploy and scale their services independently of all of the other teams.
- Each microservice is relatively small
- Comfortable for a developer to understand
- The IDE is faster making developers more productive
- The application starts faster, which makes developers more productive, and speeds up deployments
- Improved fault isolation. For example, if there is a memory leak in one service then only that service is affected. The other services continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- Microservices Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack. Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

Drawbacks:

- Developers must deal with the additional complexity of creating a distributed system.
- Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.
- Testing is more difficult as compared to Monolith applications.
- Developers must implement the inter-service communication mechanism.
- Implementing use cases that span multiple services without using distributed transactions is difficult.
- Implementing use cases that span multiple services requires careful coordination between the teams.
- Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different service types.
- Increased memory consumption. The microservice architecture replaces N monolithic application instances with NxM services instances. If each service runs in its **Container**, which is usually necessary to isolate the instances, then there is the overhead of M times as many Containers.



6.2 TYPES OF DATABASES ON AZURE

Azure offers a choice of fully managed relational, NoSQL and in-memory databases, spanning proprietary and open-source engines, to fit the needs of modern app developers. Infrastructure management—including scalability, availability and security—is automated, saving you time and money. Focus on building applications while Azure managed databases make your job simpler by surfacing performance insights through embedded intelligence, scaling without limits and managing security threats.

6.2.1 Find the database product you need

IF YOU WANT TO	USE THIS
Managed, intelligent SQL in the cloud	Azure SQL Database
Managed, always up-to-date SQL instance in the cloud	Azure SQL Managed Instance
Migrate your SQL workloads to Azure while maintaining complete SQL Server compatibility and operating system-level access	SQL Server on Virtual Machines
Build scalable, secure and fully managed enterprise-ready apps on open-source PostgreSQL, scale out single-node PostgreSQL with high performance or migrate PostgreSQL and Oracle workloads to the cloud	Azure Database for PostgreSQL
Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MySQL database service or migrate MySQL workloads to the cloud	Azure Database for MySQL
Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MariaDB database service	Azure Database for MariaDB
Build applications with guaranteed low latency	Azure Cosmos

and high availability anywhere, at any scale or migrate Cassandra, MongoDB and other NoSQL workloads to the cloud	DB
Power fast, scalable applications with an open-source-compatible in-memory data store	Azure Cache for Redis
Accelerate your transition to the cloud using a simple, self-guided migration process	Azure Database Migration Service
Modernize existing Cassandra data clusters and apps, and enjoy flexibility and freedom with managed instance service	Azure Managed Instance for Apache Cassandra

Not available	Azure SQL Database	Azure SQL Managed Instance	SQL Server on Virtual Machines	Azure Database for PostgreSQL	Azure Database for MySQL	Azure Database for MariaDB	Azure Cosmos DB	Azure Cache for Redis
Relational Database	Available	Available	Available	Available	Available	Available	Not available	Not available
Non-Relational Database (NoSQL)	Not available	Not available	Not available	Not available	Not available	Not available	Available	Not available
In-Memory Database	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Available
Data Models	Relational	Relational	Relational	Relational	Relational	Relational	Multi-Model: Document Wide-column Key-Value Graph	Key-Value
Hybrid	Available	Available	Available	Available *(Hyperscale)*	Not available	Not available	Not available	Not available
Serverless Compute	Available	Not available	Not available	Not available	Not available	Not available	Available	Not available

Storage Scale Out	Available *(Hyperscale)*	Not available	Not available	Available *(Hyperscale)*	Not available	Not available	Available	Available
Compute Scale Out	Available *(Hyperscale - read-only)*	Not available	Not available	Available *(Hyperscale)*	Not available	Not available	Available	Available
Distributed Multi-Master Writes (Write data to different regions)	Not available	Not available	Not available	Not available	Not available	Not available	Available	Available *(Coming Soon)*
OSS Based Service (Community edition and open extension support)	Not available	Not available	Not available	Available	Available	Available	Not available	Available
HTAP (Available with Azure Synapse Link)	Available *(Coming Soon)*	Not available	Not available	Available *(Coming Soon)*	Not available	Not available		

6.3 AZURE STACK

Microsoft Azure Stack is a true hybrid cloud computing solution which is an extension of Azure allowing organizations to provide Azure services using their own on-premises data centers. The data centers convert into a Public Cloud that runs Microsoft's Azure Cloud platform. The basic principle of the Azure Stack is to enable organizations to hold sensitive data and information with their own data centers while giving the ability to reach the public cloud of Azure. Similar to Azure, Azure Stack services run on top of Microsoft Hyper-V on Windows and uses Microsoft's Networking and storage solution to function seamlessly.

The Microsoft Azure Stack is an appliance built only for specific server vendor's hardware and is distributed by those vendors to bring the Azure Cloud to organization's on-premises data centers. Today, most of the major hardware vendors such as Dell, Lenovo, HP, Cisco, and Huawei support the Azure Stack with many more vendors getting approved regularly.

Figure 2 Source: Microsoft Azure Stack Vendors

6.4 PURPOSE OF AZURE STACK

Most modern-day organization's mandatory cloud requirement is to deliver IT power, services, and resources that are flexible, highly scalable and at the same time extremely cost effective. Implementing such a cloud-adaptive environment requires a high start-up cost to implement which also brings many challenges. On the other hand, organizations that adopt the public cloud such as Azure or AWS to overcome these problems also face difficulties of migrating the workload seamlessly between the on-premises environment and the cloud.

In the past, organizations overcame such scenarios by creating a private cloud that connected to the public cloud, but these private clouds require local development, configuration, and maintenance of a complicated collection of software stacks. It makes the local data center much more complex not guaranteeing that the system's local software stacks are compatible with the public and private clouds as well as accessing and managing the data.

Microsoft Azure Stack can be implemented to overcome these challenges. The Azure Stack platform seamlessly integrates with the Azure environment extending to the local data center. It provides the consistency required by developers to build and deploy a single application for both the public and private cloud without building separate applications for each platform.

The Microsoft Azure Stack compromises with a wide variety of Azure services that can be hosted on the on-premises data center such as Azure App Services, Azure Virtual Machines, Azure Functions, and also provides services like Azure Active Directory to manage Azure Stack Identities.

6.5 BENEFITS OF AZURE STACK

Azure Stack along with Azure provides a variety of benefits such as:

Consistent Application Development

The application developers can maximize productivity as there is no need to develop separate applications for the public and private clouds since the same DevOps approach is followed for the hybrid cloud environment. This allows Azure Stack customers to

- Use powerful automation tools such as Azure PowerShell extensions
- Embrace modern open-source tools and visual studio to make advanced intelligent business applications
- Rapidly build, deploy, and operate cloud designed applications that are portable and consistent among hybrid cloud environments.
- Program in any programming language like Java, Python, Node.js, PHP, and even use open-source application platforms.

6.6 AZURE SERVICES FOR ON-PREMISES

With Azure Services availability for on-premises, businesses can adopt hybrid cloud computing and meet the businesses' technical requirements with the flexibility to choose the correct deployment that suits the business. These Azure services provide:

- Azure IaaS services beyond any traditional virtualizations, such as the use of VM scale sets that enables rapid deployments with flexible scaling sets to run modern and complex workloads.
- Azure PaaS services to run highly productive Azure app services and Azure functions in on-premises data centers.
- Common operational practices between Azure and Azure Stack, therefore, no additional skills are needed to use the Azure stack environment to easily deploy and operate Azure IaaS and PaaS services as they function on Azure.
- Build future-proof applications as Microsoft delivers innovative services to the Azure Stack like Azure Marketplace applications within the Azure Stack.

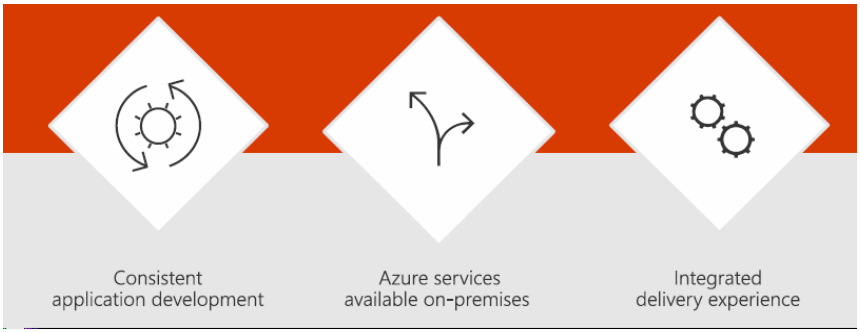


Figure 6 Source: Microsoft Azure Bootcamp 2018

Continuous Innovation

The Azure Stack is designed from the ground up to be consistent with Azure. The Azure Stack has frequent updates to the platform meaning that Microsoft prioritizes new features based on customer and business needs and delivers those requirements as soon as possible.

Updates

There are two types of updates for the Azure Stack;

1. Azure Capabilities to Azure Stack – Updates are released as soon as they are ready and typically aren't scheduled regularly. These include marketplace content and updates to existing Azure Services that are deployed on the Azure Stack
2. Azure Stack Infrastructure – Updates are released at regular time intervals since these includes firmware, drivers, etc. The infrastructure updates are usually to improve the operational excellence of the Azure Stack.

6.7 AZURE STACK IAAS

The value in Azure Stack providing cloud-native capabilities to their data enters. They see the opportunity to modernize their apps and address the unique solutions Azure Stack can deliver, but they often pause as they ponder where to begin. They wonder how to get value from the investments they have in apps currently running on virtual machines (VM). They wonder, "Does Azure Stack help me here? What if I am not quite ready for Platform-as-a-Service?" These questions are difficult, but the answers become more clear when they understand that Azure Stack at its core is an IaaS platform.

Azure Stack allows customers to run their own instance of Azure in their data center. Organizations pick Azure Stack as part of their cloud strategy because it helps them handle situations when the public cloud won't work for them. The three most common reasons use Azure Stack are because of poor network connectivity to the public cloud, regulatory or contractual requirements, or backend systems that cannot be exposed to the Internet.

Azure Stack has created a lot of excitement around new hybrid application patterns, consistent Azure APIs to simplify DevOps practices and processes, the extensive Azure ecosystem available through the Marketplace, and the option to run Azure PaaS Services

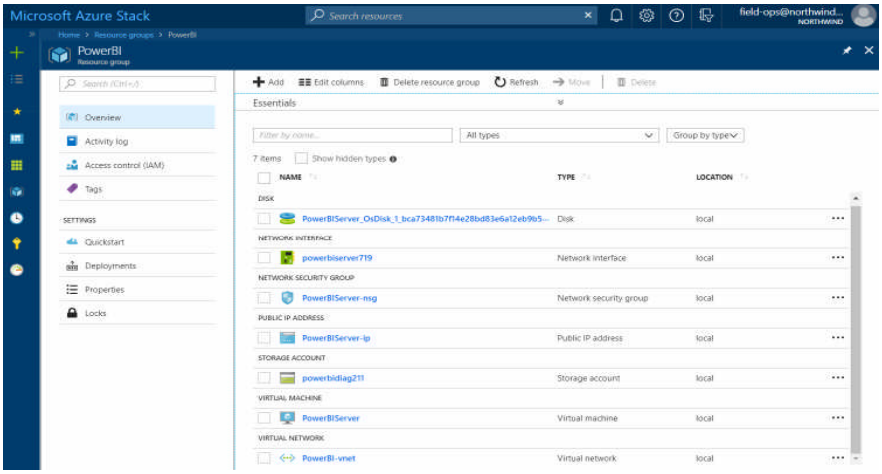
locally, such as App Services and IoT Hub. Underlying all of these are some exciting IaaS capabilities and we are so exciting to be kicking off a new blog series to show it off.

IaaS is more than virtual machines

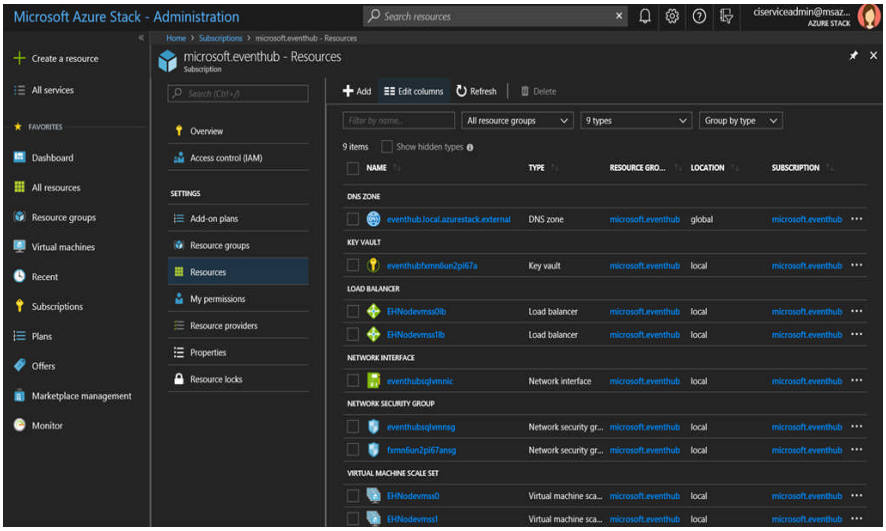
People often think of IaaS as simply virtual machines, but IaaS is more. When you deploy a VM in Azure or Azure Stack, the machine comes with a software defined network including DNS, public IPs, firewall rules (also called network security groups), and many other capabilities. The VM deployment also creates disks for your VMs on software defined storage running in Blob Storage. In the Azure Stack portal image, you can see how this full software defined infrastructure is displayed after you have deployed a VM:

IaaS is the foundation for PaaS Services

Did you know that the Azure PaaS services are powered by IaaS VMs behind the scenes? As a user you don't see these VMs, but they deliver the capabilities like Event Hubs or Azure Kubernetes Service (AKS). This same Azure IaaS is the foundation of PaaS in Azure Stack. Not only can you use it to deliver your applications, Azure PaaS services will use IaaS VMs to deliver solutions on Azure Stack.



Take Event Hubs, currently in private preview, as an example. An Azure Stack administrator downloads the Event Hubs resource provider from the Marketplace and installs it. Installation creates a new admin subscription and a set of IaaS resources. The administrator sees things like virtual networks, DNS zones, and virtual machine scale sets in the administration portal:



SUMMARY

A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece. ... In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.

Microsoft Azure Stack is a true hybrid cloud computing solution which is an extension of Azure allowing organizations to provide Azure services using their own on-premises data centers. The data centers convert into a Public Cloud that runs Microsoft's Azure Cloud platform.

QUESTION BANK

1. Explain Monolithic Architecture
2. What is Example for Monolithic Approach?
3. What are the Benefits and Drawbacks of Monolithic Architecture?
4. Explain the Microservices Architecture?
5. What are the different Types of Databases on Azure?
6. What is Azure Stack?
7. What is the Purpose of Azure Stack?
8. What are the Benefits of Azure Stack?
9. What are the Azure Services for On-Premises?
10. What is Azure Stack IaaS?

REFERENCE

1) https://download.microsoft.com/DOWNLOAD/2/C/A/2CA4DC8E-021C-4D56-8529-DF4F71FF4A1B/9780735697225.PDF
2) https://ptgmedia.pearsoncmg.com/images/9780735697225/samplepages/9780735697225.pdf
3) https://dzone.com/articles/monolithic-vs-microservice-architecture
4) https://www.sherweb.com/blog/cloud-server/microsoft-azure-stack/

❖❖❖❖

Unit III

7

NET DEVOPS FOR AZURE

Unit Structure :

7.0 OBJECTIVE

After going through this chapter, you are able to understand

- DevOp (Development Operation) Concept and its process
- DevOp Lifecycle process
- Reference Architecture of DevOps and process involves in it.
- Components of DevOps and customer feedback mechanism
- DevOps Challenges and Problems and its solutions

7.1 WHAT IS DEVOPS?

DevOps is not a technology or tool, it is a concept of behaviour, and it is an extension of Agile Methodology DevOps (Development Operations) means an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance department collaborate to deliver software in continuous manner to grasp the business opportunities in markets and collect feedback from customer from time to time. .

The DevOps is a set of practices designed to overcome the gap between development, QA and Operations by effective communication and collaboration, incorporating continuous integration process with automated deployment. DevOps helps to increase an organization's speed to deliver applications and services. It allows organizations to serve their customers better and compete more strongly in the market. There are four basic continuous processes in DevOps:

- Continuous Integration
- Continuous Delivery
- Continuous Testing
- Continuous Monitoring

7.1 DEVOPS LIFECYCLE

DevOps is deep integration between development and operations. Understanding DevOps is not possible without knowing DevOps life cycle. Here is brief information about the Continuous DevOps life-cycle:

7.1.1 Development*In this DevOps stage the development of software takes place constantly. In this phase, the entire development process is separated into small development cycles. This benefits DevOps team to speed up software development and delivery process.*

7.1.2*Testing* *QA team use tools like Selenium to identify and fix bugs in the new piece of code.*

7.1.3*Integration* *In this stage, new functionality is integrated with the prevailing code, and testing takes place. Continuous development is only possible due to continuous integration and testing.*

7.1.4*Deployment* *In this phase, the deployment process takes place continuously. It is performed in such a manner that any changes made any time in the code, should not affect the functioning of high traffic application.*

7.1.5*Monitoring* *In this phase, operation team will take care of the inappropriate system behavior or bugs which are found in production.*
In short we can say that Lifecycle of DevOps perform following operation

- The Dev writing code
- Building & deploying binaries in a QA environment
- Executing test cases and finally
- Deploying onto Production in one smooth integrated flow.

Obviously, this approach places a great emphasis on automation of Build, Deployment, and Testing. Use of Continuous Integration (CI) tools, Automation Testing tools become a norm in a DevOps cycle

7.2 REFERENCE ARCHITECTURE OF DEVOPS

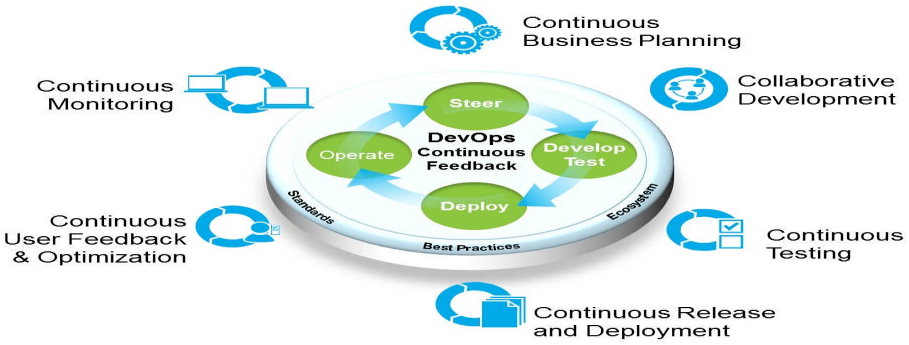


Fig. 7.1 DevOps Architecture (Reference from IBM's DevOps Dummies)

The DevOps reference architecture shown in above Figure 2 proposes the following four sets of adoption paths:

i. Plan
ii. Develop/Test
iii. Deploy
iv. Operate

7.2.1Plan

This adoption path consists of one practice that focuses on establishing business goals and adjusting them based on customer feedback: *continuous business planning*. Businesses today need to be

agile and able to react quickly to customer feedback. Achieving this goal centers on an organization's ability to do things right. Information required to plan and re-plan quickly, while maximizing the ability to deliver value, is fragmented and inconsistent.

7.2.2 Develop/Test

This adoption path involves two practices: collaborative development and continuous testing. As such, it forms the core of development and quality assurance (QA) capabilities. *Collaborative development* enables these practitioners to work together by providing a common set of practices and a common platform they can use to create and deliver software. For ex. software developers continuously or frequently integrate their work with that of other members of the development team with following advantages

1. Enable ongoing testing and verification of code
2. Validate that the code produced and integrated with that of other developers and other components of the application functions and performs as designed
3. Continuously test the application being developed

Continuous testing means testing earlier and continuously across the life cycle, which results in reduced costs, shortened testing cycles, and achieved continuous feedback on quality. This process is also known as *shift-left testing*, which stresses integrating development and testing activities to ensure quality is built-in as early in the life cycle as possible and not something left to later.

7.2.3 Deploy

The Deploy adoption path is where most of the root capabilities of DevOps originated. Continuous release and deployment take the concept of continuous integration to the next step. This facilitates continuous deployment of software to QA and then to production in an efficient, automated manner. The goal of continuous release and deployment is to release new features to customers and users as soon as possible.

7.2.4Operate

The Operate ad option path includes two practices that allow businesses to monitor howre leased applications are performing in production and to receive feedback from customers.

This data allows the businesses to reactinanagile manner and change their business plans as necessary.

In a way, you may argue that to implement DevOps you need tools. It is true but tools are only accelerators. But actually, it is about the following 3 aspects:

People: It is very important to train and have a highly motivated team of people to be able to effectively communicate and collaborate through this entire journey of cultural change.

Process: As we are talking about cultural change for DevOps implementation it is very much a necessity to have practices and strategies which provide value to the customer. A proper way of doing it would be to do an AS-IS maturity assessment, look at gaps and propose a roadmap for implementation of giving appropriate recommendations.

I will not be talking in-depth about how I have got about doing these assessments but I will be glad to share any inputs on the same.

Tools: Finally, it is about using the accelerators by automating the process using standard DevOps tools that are available today. It could be Open-Source (Jenkins, Git etc.), Commercial (Microsoft TFS, VSTS, IBM Rational, Jira etc.) or a mix of both.

7.3 COMPONENTS OF DEVOPS

Let's now look at the following 4 components of DevOps which form the core from an implementation point of view and also the organizations have developed good automation frameworks around the same offering it as a service to their clients.

- Continuous Integration
- Continuous Testing
- Continuous Delivery
- Continuous Monitoring

I have truly believed that if a developer has to work in this mode then there should be an execution item like a Task or a Defect (In Agile it can be a part of User Story) assigned to him to enable him to deliver the work within the sprint timeframe. Let's now take a look at each of these components in detail.

7.3.1 *Continuous Integration*

As a developer, you work on the tasks or defects assigned to and check-in the code to a shared repository multiple times in a day. Similarly the other members of the team also check-in the code to the shared repository. You will then actually integrate all the work done by the team members in a common build server and perform an automated build. Doing these integrations and automated builds on a regular basis is called Continuous Integration. This practice helps to detect issues very early and also ensures that all the modules which are integrated work as required. So if you do not follow this approach then the integration of the team's work may happen once in a month which may be late to find and fix any integration issues.

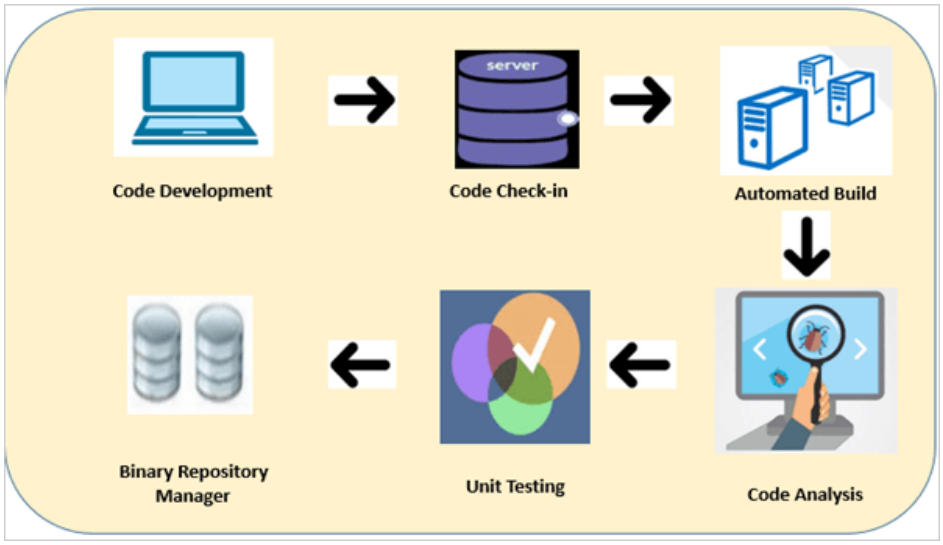


Fig.7.2 Sample Continuous Integration workflow

7.3.2 *Continuous Delivery*

Continuous Delivery is the next step after Continuous integration. The goal of Continuous Delivery is to push the application built into production as quickly as possible. During this process, it goes through various stages in the lifecycle of delivery i.e. QA, Staging, Production environments etc. This process of regularly delivering the applications built into various stages is known as Continuous Delivery.Continuous delivery helps in quicker time to market when compared to traditional methods, lesser risk, lowering the cost by encouraging more automation in the release process and most importantly getting faster feedback from the end users to produce a quality product.

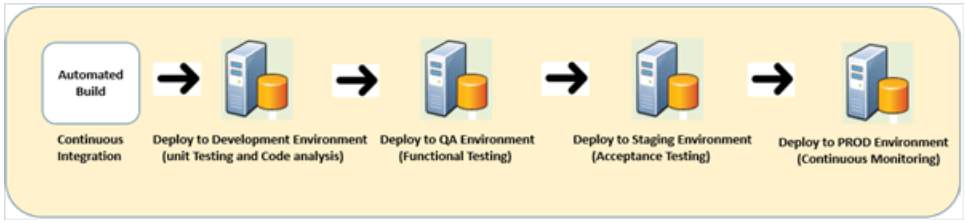


Fig.7.3 Sample Continuous Delivery Workflow

In the above diagram, you can look at different environments available and so this provisioning of the infrastructure for the environments can also be automated during this continuous delivery process.

7.3.3 *Continuous Testing*

From the above 2 practices, we came to know that CI and CD help to deploy the application or changes to the production. This whole process involves proper validation of code and its integration with all the components involved in it to ensure that the application works as envisaged and is free of bugs or defects. So Continuous Testing is the process of running various types of automated tests starting with CI

process till the time the application is finally deployed to production. You can see from the previous diagram that in the Continuous Integration step we integrate all of the developers work into a common build server and also during this stage the developers would run a certain amount of unit tests.

Once these integration and tests work without any errors, only then the application or changes are deployed to the QA environment after applying for these quality gates and approvals. In the QA environment, the functional tests are run and again based on the approvals it would be deployed to staging environment which would be on parity like the production systems and acceptance tests run. Once this activity is completed the application or the changes are finally deployed into the production systems.

So one can note here that continuous testing as an activity starts from the CI stage itself and is a very mandatory step throughout the continuous delivery process.

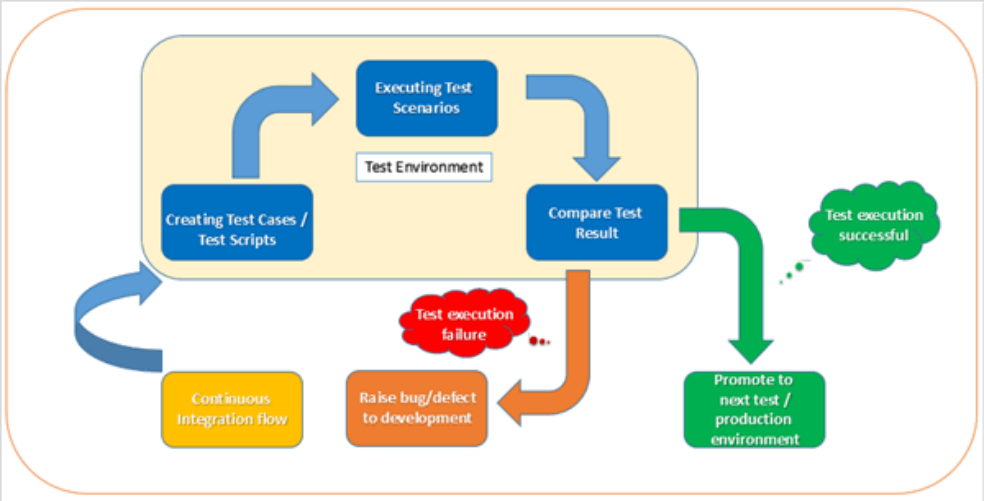


Fig. 7.4 **Sample Testing workflow in the continuous delivery process**

7.3.4 *Continuous Monitoring*

As the application or changes are deployed to the production environment the operations team will look to monitor the application and environment from an up-time, stability, availability point of view. This process is known as Continuous monitoring. The operations teams will have their own software's to monitor the environment but will also need to play their part to monitor the applications deployed for any issues. For this, they would need to work with the development teams in order to build certain tools for analyzing the application issues.

So infrastructure, environment, and applications issues are all that monitored in the process of continuous monitoring. *Continuous monitoring* provides data and metrics to operations, QA, development,

lines-of-business personnel, and other stake- holders about applications at different stages of the delivery cycle.

7.4 CONTINUOUS CUSTOMER FEEDBACK AND OPTIMIZATION

The two most important types of information that a software delivery team can get are data about how customers use the application and feedback that those customers provide upon using the application. New technologies allow businesses to capture customer behavior and customer pain points right as they use the application. This feedback allows different stakeholders to take appropriate actions to improve the applications and enhance customer experience. Lines of business may adjust their business plans, development may adjust the capabilities it delivers, and operations may enhance the environment in which the application is deployed. This con- tenuous feedback loopisan essential component of DevOps, allowing businesses to be moreagile and responsive to customer needs.

7.5 DEVOPS CHALLENGES AND PROBLEMS

DevOps is a structured way of approaching software development. Today, more and more IT organizations realize its benefits as their irregular release cycles get replaced by streamlined continuous integration and continuous delivery pipelines. The advancements in customer-centric service delivery models demand streamlined business processes and improved teams' culture. DevOps helps automate the processes between development and IT support functions. And, though, the journey from the traditional, siloed IT workflows into a collaborative pipeline often offers setbacks and failures. However, by overcoming challenges and leveraging opportunities, DevOps organizations can move much faster than others.

DevOps has become mainstream, and with this comes the most basic and evident questions, that are, what challenges will I face adopting it? Despite the tremendous popularity of **DevOps** today, the myriad surveys conducted show lack of awareness, knowledge, and governance when it comes to establishing brand new and fresh application environments. Some of the major challenges faced by the organizations in **DevOps adoption** are as follows:

7.5.1 *Work Culture Shift*

DevOps implementation leads to a large workplace culture transformation. This is one of the toughest challenges that need to be dealt with as the culture of the organization gets imbibed within the employees of the place. Changing the culture of a particular place is a long-term process. In this case, the focus should be on building a collaborative

culture. The employees who are pro-DevOps have to instill and convince the concepts and benefits of this culture among the rest of the co-workers.

7.5.2 *Switch From Legacy Infrastructure To Microservices*

There has to be a replacement of older, monolithic infrastructure as it can spell stability issues. Making use of infrastructure-as-code along with **microservices** will open the floodgates to quicker development along with exquisite innovations. If your company is restricted to let changes in, then it will soon be replaced, no matter what reputation it used to hold. However, making a shift to microservices has its own set of problems as it needs to have a foundation of automation, configure-management, and continuous delivery to be able to manage additional operational workloads which microservices brings in with itself.

7.5.3 *Tool Issues*

While switching to **DevOps**, people are tempted to make use of the myriad tools available. The members become too much dependant on the tools, even if they want to cater to the smallest issue. Due to momentary attractive features, the organizations become addicted to tools with short-term benefits instead of the ones with long-term benefits. However, if they are not properly trained on the usage of a newly introduced tool, it can lead to confusion amongst team members. Some of the tools are a Saa S-based or open source and can be very easily adopted without any authorization and may prove to be harmful to the organization's health. So, teams should be given a library of tools from which they can select their preferred tools. This will also keep the leaders well-informed about their activities.

7.5.4 *Different Standards And Metrics*

The Dev and the **Ops** departments separately have different tool sets and metrics as they have totally different goals and working systems. The collaboration of these two teams can prove to be very ambiguous and inefficient. It becomes monotonous to sit together and integrate the tools, hence, it can be a very tedious task. Both the teams should agree upon a unanimously-decided metric system.

7.5.5*Process-oriented Challenges*

For the kind of people who are used to following rules blindly, it might be pretty challenging for them to adopt **DevOps** since there is no fixed framework which will assign the particular employee on how he/she should progress with a project to achieve the desired goal. The teams have to take their own call to decide the course of action. It is mostly in the non-structured format. Basically, there are no central **DevOps teams** operating which can choose the right tools and systems for the team or a particular individual. Undoubtedly, such a structure gives employees a lot of scope and space for innovation and exhibiting individual responsibility. But at the same time, it can prove to be very challenging. Different practice methods may lead to ambiguity amongst the team members.

These are the most basic challenges which every organization is bound to face while they are preparing to make a switch to **DevOps**

It doesn't matter whether you are in Cloud, Enterprise or Mobile. For each one of you, stable software delivery on time is the key to your business success. Some of the serious issues blocking your software delivery are:

- **Building and maintaining servers** – Time consuming and unproductive
- **No environment management** – Differences in development and production environments
- **Deployments are a blocker** – Upgrade risk due to manual management of multiple application configurations and versions – Dependency on specific deployment engineer
- **Hacking** – Fixing directly in production (instead of a proper hotfix process) and forgets to check-in into source control

7.6 SOLUTION OF PROBLEMS ON DEVOPS

7.6.1 *Replace Legacy Infrastructure With IaaS and Microservices.*

Utilizing outsourced Infrastructure-as-a-Service (IaaS) for the latest hardware, server and data storage solutions is a cost-effective and efficient way to propel your business processes into the 21st century. Microservices can make your application much easier to build and — even more importantly — easy to scale.

7.6.*2 Invest in A Bigger Development Team and Enhanced Security Practices.*

To quote the classic line from the Jaws movie, "You're going to need a bigger boat." If you expect to keep up with your competitors and build next-generation solutions, you have to invest in experienced developers who are knowledgeable of all the latest advancements in technology. The good news is that it is easier than ever to grow your team quickly and cost-effectively, thanks to the ability to hire dedicated development teams that can be scaled up or down on demand.

Another area that is worthy of additional investment is security. With so many stories in the news day after day of big name corporations experiencing major security breaches, developing an application with bullet-proof protection can be the one decision that causes a user to choose your solution over a competitor's.

7.6.*3 Develop a Culture of Collaboration and Transparency*

It's important to change the mindset from individual developers "doing their part" to one cohesive team that is responsible as a collective whole for the outcome of the product. All members of your DevOps team should know exactly what the others are doing at all times.

Communication and transparency is key. This can be achieved through communication tools such as Slack or simply with daily stand-up meetings.

DevOps can be truly successful only when there is a clear focus on and dedication to teamwork excellence as opposed to individual success.

7.7 SOFTWARE TOOLS FOR DEVOPS

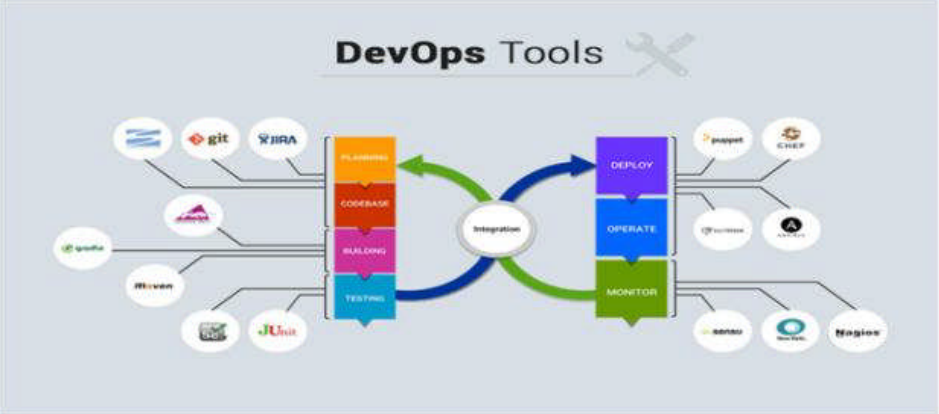


Fig. 7.5 Software Tools for DevOps

As DevOps is the collaboration of Development, QA and Operations, it is obvious that a single tool cannot be adequate for all the needs. So there are multiple tools required in each stage to perform all the operations successfully.

Popular Tool for DevOps Automation:

- Git : Version Control System tool
- Jenkins : Continuous Integration tool
- Selenium : Continuous Testing tool
- Puppet, Chef, Ansible : Configuration Management and Deployment tools
- Nagios : Continuous Monitoring tool
- Docker : Containerization tool

How do all these tools work together?

This flow may vary from organization to organization as per the requirement.

- Developers develop the code and this source code is managed by Version Control System tools like Git etc.
- Developers send this code to the Git repository and any changes made in the code is committed to this Repository.
- Jenkins pulls this code from the repository using the Git plugin and builds it using tools like Ant or Maven.

- Configuration management tools like puppet deploys & provisions testing environment and then Jenkins releases this code on the test environment on which testing is done using tools like selenium.
- Once the code is tested, Jenkins send it for deployment on the production server (even production server is provisioned & maintained by tools like puppet).
- After deployment It is continuously monitored by tools like Nagios.
- Docker containers provides testing environment to test the build features

SUMMARY:

DevOps is not just about tools but it also includes a set of best practices that enables to bridge the gap between the development and operations teams in the areas of continuous integration and deployment by using an integrated set of tools to automate the software delivery. So the goal of DevOps is simply to help any organization in the speed of delivering applications to the end-users and enabling faster end-user feedback which is the need for any business today.

REFERENCE

- **DevOps For Dummies ® , 2nd IBM Limited Edition**

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

- ***DevOps for Azure Applications***

Suren Machiraju
Suraj Gaurav
Apress Publication

QUESTION FOR SELF-STUDY

Q1. What are the different DevOps tools available in market. Explain one tool working

Q2. What do you mean by Microsoft Azure Platform.
Q3. What are the different components of DevOps. Explain each in brief.

8

MULTIPLE PROJECT TRACKING TEMPLATES

Unit Structure

8.0 OBJECTIVE

After reading this chapter, you will read following concept.

- Various in-built Project Tracking Templates use in Development Operations.
- Working with DevOps Tracking Process and its methodologies

In an active organization, you will likely manage multiple projects (at different stages of completion) simultaneously. Download any of the following free, customizable templates, available in Excel, Google Sheets, and Smartsheet formats, to track the details of more than one project at a time.

8.1 MULTIPLE PROJECT TRACKING TEMPLATE

MULTIPLE PROJECT TRACKING TEMPLATE

				PROJECTS			DELIVERABLE(S)		COST / HOURS		
AT RISK	STATUS	PRIORITY	DEADLINE	TASK	DESCRIPTION	ASSIGNED TO	DELIVERABLE	% DONE	FIXED COST	ESTIMATED HOURS	ACTUAL HOURS
☐				PROJECT NAME							
☐	Complete	High		Task 1				100%			
☑	In Progress	High		Task 2				80%			
☑	Overdue	High		Task 3				30%			
☐	Not Started	Medium		Task 4				0%			
☐	On Hold	Low		Task 5				10%			
☐				PROJECT NAME							
☐				Task 1							
☐				Task 2							
☐				Task 3							
☐				Task 4							
☐				PROJECT NAME							
☐				Task 1							
☐				Task 2							
☐				Task 3							
☐				Task 4							
☐				Task 5							
☐											
☐											
☐											
☐											
☐											
☐											
☐											
☐											

Fig. 8.1

Quickly gain an overview of task status across multiple projects. This template highlights status, priority, and task deadlines, and whether or not items are at risk. You can also track the percentage of tasks completed, fixed costs, and estimated and actual hours spent. If you want, you can also add a column for billed hours

MULTIPLE PROJECT BUDGETING TEMPLATE

						LABOR			MATERIALS			OTHER							BALANCE
TASK ID	DESCRIPTION	STATUS	START DATE PLANNED	START DATE ACTUAL	END DATE	HR	$ / HR	LABOR TOTAL	UNITS	$ / UNITS	MATERIALS TOTAL	TRAVEL	EQUIPMENT / SPACE	FIXED	MISC.	OTHER TOTAL	BUDGET	ACTUAL	UNDER / OVER
	PROJECT																		
	Task	Complete				35.0	$ 40.00	$ 1,400.00	25	$ 5.00	$ 125.00	$ 500.00	$ 1,000.00	$ 50.00		$ 1,550.00	$ 900.00	$ 3,075.00	$ (2,175.00)
	Task	In Progress				10.0	$ 50.00	$ 500.00	1	$ 350.00	$ 350.00			$ 100.00		$ 100.00	$ 750.00	$ 950.00	$ (200.00)
	Task	Overdue				15.0	$ 25.00	$ 375.00	300	$ 11.00	$ 3,300.00					$ -	$ 2,700.00	$ 3,675.00	$ (975.00)
	Subtask	Not Started						$ -			$ -					$ -		$ -	$ -
	Subtask	On Hold						$ -			$ -					$ -		$ -	$ -
	Subtask	In Progress						$ -			$ -					$ -		$ -	$ -
	Subtask	Overdue						$ -			$ -					$ -		$ -	$ -
	Task	Not Started						$ -			$ -					$ -		$ -	$ -
	Task	Not Started						$ -			$ -					$ -		$ -	$ -
								$ 2,275.00			$ 3,775.00	$ 500.00	$ 1,000.00	$ 150.00	$ -	$ 1,650.00	$ 4,350.00	$ 7,700.00	$ (3,350.00)

						LABOR			MATERIALS			OTHER							BALANCE
TASK ID	DESCRIPTION	STATUS	START DATE PLANNED	START DATE ACTUAL	END DATE	HR	$ / HR	LABOR TOTAL	UNITS	$ / UNITS	MATERIALS TOTAL	TRAVEL	EQUIPMENT / SPACE	FIXED	MISC.	OTHER TOTAL	BUDGET	ACTUAL	UNDER / OVER
	PROJECT																		
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -					$ -		$ -	$ -
								$ -			$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -	$ -

Fig. 8.2

Manage the budget of multiple projects with this template. Leverage the color-coding to see task status at a glance. Track labor and materials costs, travel and office expenses, and compare your planned budget vs. your actual spend. The template automatically indicates if you are under or over budget.

a.Multiple Project Task Tracking Template

PROJECT TASK TEMPLATE

PROJECTS						DELIVERABLE(S)		COST/HOURS		
STATUS	PRIORITY	DEADLINE	TASK	ASSIGNEE	DESCRIPTION	DELIVERABLE	% DONE	FIXED COST	ESTIMATED COST	ACTUAL HOURS
			PROJECT NAME							
☒	HIGH	8/27/22	Task name		Details of task here		100%	$ -		
☐		8/27/22	Task				50%	$ -		
☐		12/31/22	Task				0%	$ -		
☐										
			PROJECT NAME							
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		
			PROJECT NAME							
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		
			PROJECT NAME							
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		
☐							0%	$ -		

STATUS MENU
☐
☒

PRIORITY MENU
HIGH
MEDIUM
LOW

Fig. 8.3

In order to manage a project effectively, you and your team need to track the status of tasks that contribute to the project deliverables. Use this project tracking template to record the task status and priority, deadline, task owner, task description, percentage complete, and task cost. Customize the template to track as many projects as you need.

b. Color-Coded To-do List Template

TO-DO LIST TEMPLATE

TASK	STATUS	STATUS	DUE DATE	ASSIGNED TO	DELIVERABLE
Task 1	Complete	High			
Task 2	Complete	High			
Task 3	In Progress	High			
Task 4	Overdue	High			
Task 5	Not Started	Medium			
Task 6	On Hold	Low			
Task 7					
Task 8					
Task 9					
Task 10					
Task 11					
Task 12					
Task 13					
Task 14					
Task 15					
Task 16					
Task 17					
Task 18					
Task 19					
Task 20					
Task 21					
Task 22					
Task 23					
Task 24					
Task 25					

Fig. 8.4

If you have a small team or are responsible for resolving issues as part of a larger project, a to-do list or task list may be the right form for tracking work. This to-do list template provides space for the item name, the status and priority, the assignee, the due date, and the deliverable. Color-coding can help you differentiate high priority items and item status at a glance.

c. Excel Project Management Tracking Templates

Part of project management includes tracking the progress of tasks towards final deliverables. However, you may also need to monitor other aspects of project performance, such as project risks, KPIs, and overviews of work and budgets.

Project Management Dashboard Template

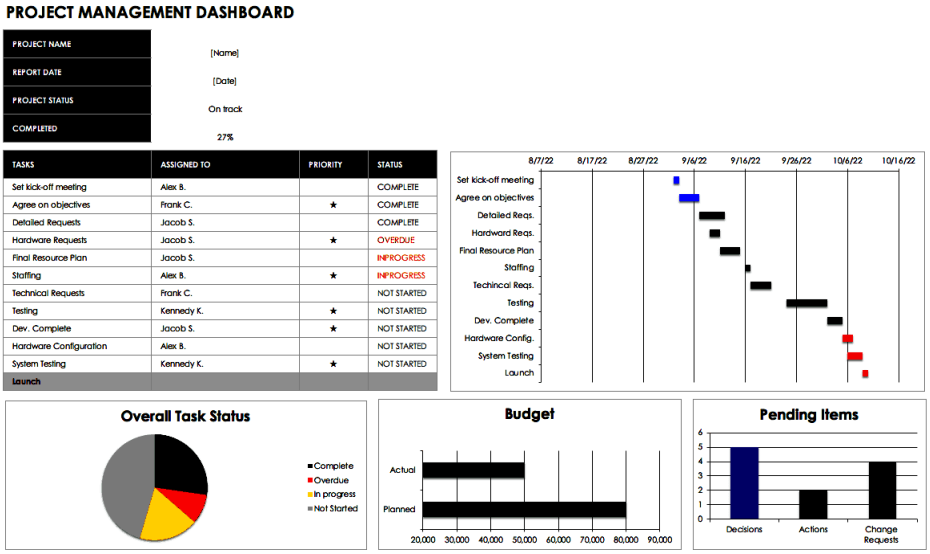


Fig. 8.5

Dashboards offer a convenient summary of activities and status not only for you, but also for your team and stakeholders. This dashboard template is ready to use out of the box: List project tasks, the assignees, the task priority, and the task status. The built-in Gantt chart updates as you change the status of each task, and graphs automatically track overall status, budget, and pending items.

d. Project Cost Tracking Templates

Costs form one of the triple constraints in project management. Use the following project budget tracking templates available in Excel, Google Sheets, and Smartsheet to stay on budget and to help anticipate any risks of exceeding the budget.

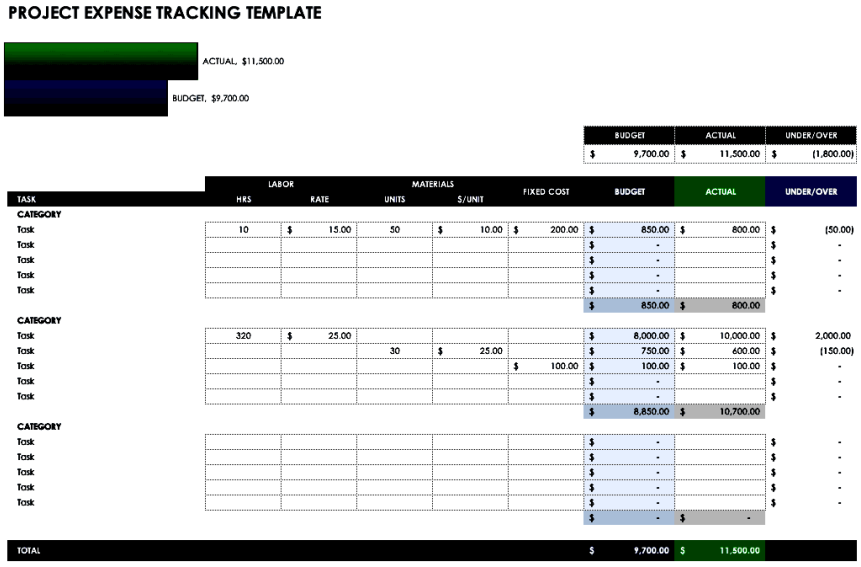


Fig. 8.6

e. Google Sheets Project Expense Tracking Template

Tracking project expenses is essential to maintaining a budget and avoiding surprise shortfalls that could eat into potential revenue or bonuses. The Google Sheets template offers a convenient online and offline format that anyone with the right permissions can access. The other formats of the template highlight planned and actual budget, and track expenses by task, noting hourly labor rates, material unit costs, and fixed rates.

f. Project Time Tracking Template

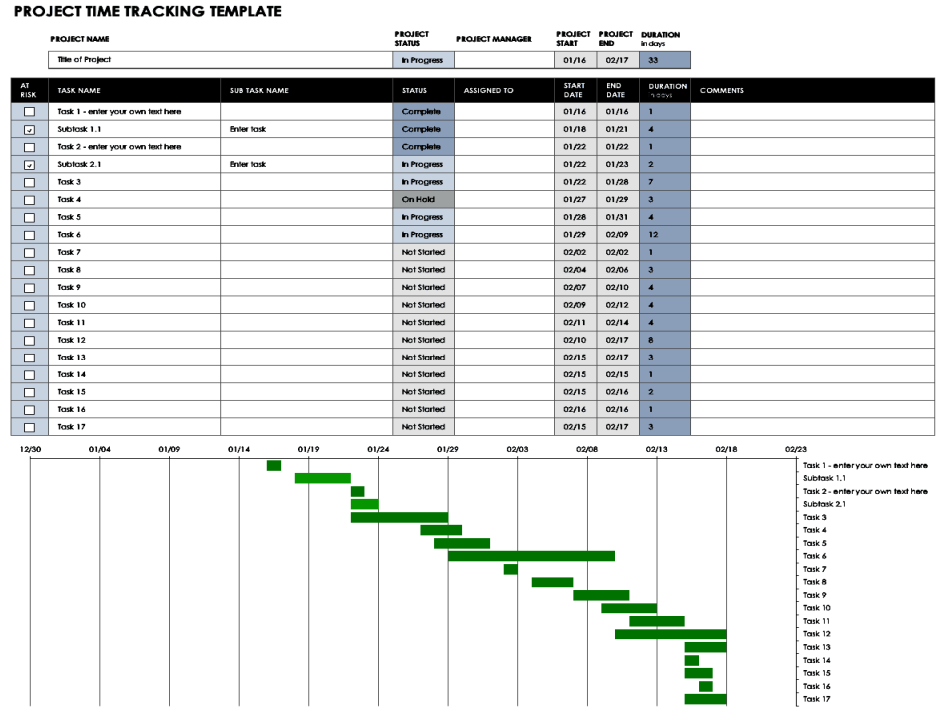


Fig. 8.7

Project timelines summarize and visualize the structure of a project. This customizable project time tracking template lists tasks and subtasks, start and end dates, task duration, and status. As you enter and update these details, changes are automatically reflected in the built-in Gantt chart.

8.2 WORKING WITH DEVOPS TRACKING PROCESS

The following points must be considered for Tracking Process.

8.2.1 Practice agile methodology

Agile project management is the first prerequisite for introducing DevOps. Following the agile methodology, teams divide work into small chunks. It allows them to deliver small but incremental features fast, which lays the basis for DevOps' continuous deployment practice.

Agile development teams implement adaptive planning and welcome changing requirements at any stage. They are flexible enough to handle unplanned work and respond to feedback coming from the operations team. From the cultural point of view, Agile and DevOps share a lot. Both of them embrace change and put the business value of the software product in priority. DevOps builds upon Agile and extends its principles beyond the development team by bringing them to the operations team.

*8.2.*2 Continuously automate processes

Automation accelerates the development cycle by reducing the amount of manual work. It helps you push code in production more frequently and produce consistent, reliable, and safe software. You need a good strategy and vision on what to automate and how because you don't want to make bad processes happen faster. You should start with automated testing to remove manual processes from unit tests, integration tests, and performance tests. Each test should and take a few minutes to run. After the testing is automated, you can build DevOps pipelines to automate the build - configure - deploy - test - release cycle. In parallel, you can introduce automation in your infrastructure configuration management and performance monitoring tasks. It will allow you to better control software that is running in production.

8.2.3 Follow CI/CD best practices

Continuous integration and deployment best practices are at the heart of the DevOps culture. Implementing them often requires the biggest investment and effort. At the same time, it has the most sufficient impact on the results if done following Martine Flower's continuous integration best practices:

- Maintain a single source repository
- Automate the build process

- Make the build self-testing
- Commit to the mainline daily
- Trigger a build after every commit
- Test in a clone of the production environment
- Make fast feedback on quality available to everyone
- Make fixing broken builds a priority task

This list can be extended by several continuous delivery best practices. By implementing those, you can make sure that your software remains in a deployable state throughout its lifecycle. It means that you can deploy your system to production on demand at any time.

- Build apps with loosely-coupled architecture
- Integrate security into the development and testing stages
- Fully automate the deployment process
- Use a version control system

When continuous delivery works well, for some projects, for example, web apps, it may evolve into continuous deployment. At this level, following continuous deployment best practices empowers your team to deploy every code change to production as soon as possible.

8.2.4 Choose your DevOps tools wisely

DevOps tools represent a set of solutions that enable the collaboration of development and operations teams across development, testing, deployment, and performance monitoring. Your toolset may include:

- Work planning and tracking (Jira, Confluence)
- Development environment (Kubernetes, Docker)
- Source control tools (Github, Gitlab, Bitbucket)
- Infrastructure provisioning IaC (Ansible, Puppet, Terraform)
- CI/CD pipelines (Jenkins, AWS Code Pipeline, CircleCI)
- Test automation, management, and orchestration (Mabl, Xray, Zephyr)
- Deployment automation (Code Deploy, Bitbucket Pipelines)
- Performance monitoring (Appdynamics, DataDog, SumoLogic)
- Change management and problem tracking (Jira Service Desk, Opsgenie)

Choosing an optimal combination of tools provided by different vendors often requires experimenting. When building your DevOps toolchain, you'll have to:

- Decide between multi-purpose and single-purpose tools
- Choose among several similar solutions by different vendors
- Select tools that integrate seamlessly with each other
- Integrate legacy tools your company uses into the new ecosystem

With the variety of DevOps tools available, it's easy to accommodate too many of them and overcomplicate your processes. To avoid this, you should prioritize processes over tools and refrain from copying someone else's automation solutions blindly.

8.2.5 Make your software observable

By giving preference to loosely coupled architectures, you ease the deployment of new functionality on your developers. However, this causes an additional burden on your operations team, which has to monitor a complex and ever-changing system.

When working with distributed systems, your operations team will have to cope with unpredicted patterns and properties. According to monitoring and observability best practices, tracking the logs, traces, and metrics is not enough anymore. You should create an own diagnostic system to identify unknown unknowns, understand what causes problems in your system and debug it quickly. For this, you should have:

- A black-box or a white-box monitoring system that allows connecting data from all points of monitoring in one place.
- A symptom-based alerting system that will notify about possible problems before they happen.
- Ability to quickly create monitoring dashboards to visualize metrics that matter at this particular time.

The performance of your monitoring system should also be measured. To know if it performs well, you should keep an eye on the number of changes made to your monitoring configuration, the adequateness of the alerts that your system sends, and the time it takes your team to solve issues.

8.2.6 Shorten feedback loops

DevOps strives to reduce wasted effort not only through process automation but also through information exchange between team members. It strives to facilitate feedback on people's work efficiency and quality at all stages of the development cycle:

- Code test results become available to developers withing minutes
- Bugs and failures are reported immediately to be fixed right away
- User feedback and usage reports are communicated for every new feature

- Performance characteristics get displayed on visual management boards
- Customer satisfaction metrics and feedback shared with the team

Sharing and implementation of customer feedback are often overlooked. This best practice emphasizes the importance of doing this to better address customer needs. This way you can make sure every new feature is solving a real problem and is appreciated by your customers.

8.2.7 Transform culture and mindsets

DevOps is a culture, not a role. Adopting DevOps practices requires creating an environment for cross-team communication and collaboration. When introducing DevOps culture, you'll work on:

- Building trust and transparency between development and operations
- Instilling an attitude of shared responsibility and ownership
- Promoting customer-centricity and empathy across teams

SUMMARY

This article defines operational and object limits placed on work tracking operations and work tracking customization. In addition to the specified hard limits on select objects, as you plan and track your project, you'll find you may want to configure a feature or customize your experience to meet your team's tracking needs. You configure teams and team. Agile tools through the web portal administration context for Azure Boards. The method you use to customize projects, which impacts all teams, depends on the process model you use.

REFERENCE:

1. **Agile Project Management with Azure DevOps: Concepts, Templates, and Metrics 1st ed. Edition,** by Joachim Rossberg**,** Publisher: Apress
2.https://docs.microsoft.com/en-us/azure/devops/organizations/settings/work/

QUESTIONS FOR SELF-STUDY

Q1. What are the different steps involve DevOps tracking process.
Q2. Explain Software Development Lifecycle in Microsoft Azure platform development.
Q3. Which point must be considered while customizing process and publish on Azure.

9

TRACKING WORK AND CODE

Unit Structure :

9.0 OBJECTIVE:

After reading this chapter, you will covered following points

- Git basic concepts and advantages
- DVCS Terminologies
- Azure DevOps Repository
- Difference between Azure DevOps vs GitHub

9.1GIT - BASIC CONCEPTS

Version Control System

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS −

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.
- Maintains a history of every version.

Following are the types of VCS −

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

In this chapter, we will concentrate only on distributed version control system and especially on Git. Git falls under distributed version control system.

Distributed Version Control System

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, distributed version control system (DVCS) comes into picture.

DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. Git does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.

9.2 ADVANTAGES OF GIT

Free and open source

Git is released under GPL's open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

Fast and small

As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

Implicit backup

The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.

Security

Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and

commit message and any other data from the Git database without knowing Git.

No need of powerful hardware

In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

Easier branching

CVCS uses cheap copy mechanism, If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.

9.3 DVCS TERMINOLOGIES

Local Repository

Every VCS tool provides a private workplace as a working copy. Developers make changes in their private workplace and after commit, these changes become a part of the repository. Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.

Working Directory and Staging Area or Index

The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn't track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.

Let us see the basic workflow of Git.

Step 1 − You modify a file from the working directory.

Step 2 − You add these files to the staging area.

Step 3 − You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.

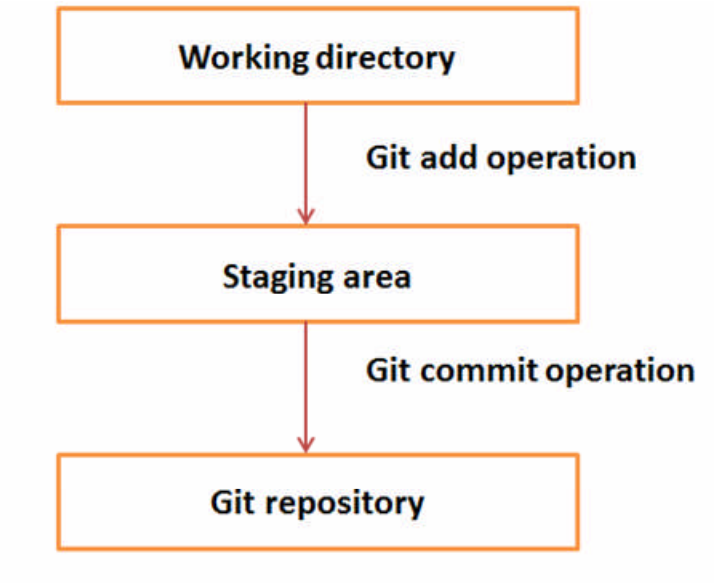


Fig. 9.1 Git Repositories Diagram

9.4 AZURE DEVOPS REPOSITORY

Azure Repository is a set of version control tools that we can use to manage our code. In case if we are entirely new to version control, then version control enables us to track changes we make in our code over time. There are so many software that is available in the market to enable version control on our code. We can use the version control system to keep track of each change done by each developer, safely merge them, test the changes, and publish the change into production.

There are two types of version control in Azure Repos.

- **Git:** It is a distributed version control.
- **Team Foundation Version Control:** It is a centralized version control.

Azure Repos Concepts

1. **Repository:** A repository is a location for our code, which is managed by version control. It supports Git and TFVC so we can create multiple repositories in a single project and various branches for each repository.
2. **Branch:** A branch is a lightweight reference that keeps a history of commits and provides a way to isolate changes for a feature or a bug fix from our master branch and other work.
3. **Branch policies:** It is an essential part of the Git workflow. We use them to help protect the critical branches is our development, as the master.
4. **Pull and Clone:** Create a complete local copy of an existing Git repo by cloning it. A pull command updates the code in our local repository with the code that is in the remote repository.

5. **Push and Commit:** A commit is a group of change saved to our local repository. We can share these changes to the remote repository by pushing.
6. **Fork:** A fork is a complete copy of a repository, including all file commits, and (optionally) branches.
7. **Git:** Git is a distributed version control system. Our local copy of code is a complete version control repository that makes it easy to work offline or remotely.
8. **Notification:** Using notification, we will receive an email whenever any changes occur to work items, code reviews, pull requests, source control files and builds.
9. **Projects:** A project provides a place where a group of people can plan, track progress, and collaborate on building software solutions.
10. **Teams:** A team corresponds to a selected set of project members. With teams, organizations can subcategorize work to better focus on all of the work they track within a project.

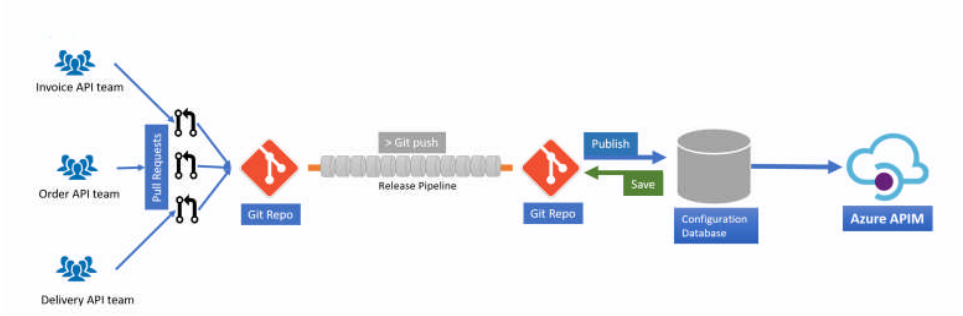


Fig.9.2 Azure Repository Diagram

9.5 AZURE DEVOPS VS GITHUB: WHAT ARE THE DIFFERENCES?

What is Azure DevOps?

Services for teams to share code, track work, and ship software. Azure DevOps provides unlimited private Git hosting, cloud build for continuous integration, agile planning, and release management for continuous delivery to the cloud and on-premises. Includes broad IDE support.

What is GitHub? *Powerful collaboration, review, and code management for open source and private development projects*. GitHub is the best place to share code with friends, co-workers, classmates, and complete strangers. Over three million people use GitHub to build amazing things together.

Azure DevOps can be classified as a tool in the **"Project Management"** category, while GitHub is grouped under **"Code Collaboration & Version Control"**.

Some of the features offered by Azure DevOps are:

- Agile Tools: Kanban boards, backlogs, scrum boards
- Reporting: dashboards, widgets, Power BI
- Git: free private repositories, pull requests

On the other hand, GitHub provides the following key features:

- Command Instructions
- Source Browser
- Git Powered Wikis

SUMMARY

- GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. Specifically, Git is a distributed version control system, which means that the entire codebase and history is available on every developer's computer, which allows for easy branching and merging. Azure Pipelines provides unlimited CI/CD minutes and 10 parallel jobs to every GitHub open source project for free. All open source projects run on the same infrastructure that our paying customers use. That means you'll have the same fast performance and high quality of service. Many of the top open source projects are already using Azure Pipelines for CI/CD, such as Atom, CPython, Pipenv, Tox, Visual Studio Code, and TypeScript etc..

QUESTIONS

Q1. In Microsoft Azure tools, how project tracking operation and testing are perform.

Q2. How you can use GitHub repository. Explain and demonstrate yourself.

Q3. Explain the Git repository functions.

REFERENCES

1. DevOps for Azure Applications Deploy Web Application on Azure By Suren Machiraja, Suraj Gaurav, Published by Apress publication
2. DevOps For Dummies ® , 2nd IBM Limited Edition By Sanjeev Sharma , Bernie Coyne, Published by John Wiley & Sons, Inc.
3. https://azure.microsoft.com/en-in/services/devops/repos/

Unit IV

10

BUILDING AND VALIDATING THE CODE

Unit Structure :

10.0 OBJECTIVE

In this chapter you will learn the following concepts:

- How to build the code
- Difference in a private and integration build
- How to configure your CI build in Azure DevOps Services
- How to validate the code

10.1 INTRODUCTION

This chapter introduces the types of builds, flow of build on feature and master branch. After we are through with this chapter, you will learn how to implement a build on Azure Pipelines for a .NET core solution. You will also learn the three of the critical defect removal methods which covers static analysis, levels of testing, concept and implementation of inspections.

10.2 STRUCTURE OF A BUILD

There are two types of Build:

1) Private Build
2) Integration Build

1) Private Build

Private Build runs only on the developer's workstation and is a tool to ensure that instant changes will not disrupt the stability of the application.

2) Integration Build

The integrated build runs on a shared server and is owned by the team. It builds code from many developers. As the branch model becomes more and more popular, integrated build have been adapted to run on feature branches and also the master branches.

10.2.1 Flow of a Build on a Feature Branch

The following figure10-1 shows the sequence of build operations that occur when working on a functional branch.

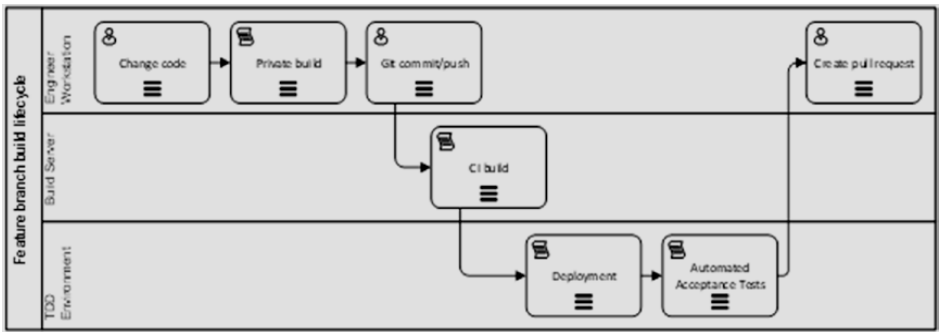


Figure 10-1: The build process for code on a feature branch flows across three environments

Explanation:

- If you change the code, your private build will run at every breakpoint to ensure your safety. If you accidentally broke something, you will learn immediately.
- Since you work in Git (a decentralized version control system), you will make a lot of short commits. This makes it very easy to undo changes.
- At your perception, run the private build locally. If you decide to push changes to the team's Git server, CI Build will recognize the changes and execute the integrated build process on the team's build server.
- If successful, the build will archive the generated artifacts, possibly in Azure Artefacts (Azure Artifacts is an extension that allows you to easily discover, install, and publish NuGet packages in Azure DevOps), (NuGetrepository).
- Then an automated deployment script will activate and install those built artifacts to an environment dedicated to the continuous integration process. The environment is known as "TDD (Test Driven Development) environment."
- The purpose of this environment is to verify that (1) the new version of the software can be installed and (2) the new version of the software still passes all its acceptance tests.
- This requires a complete system acceptance test in your code base. If you don't do this, they are easy to start developing.
- After the acceptance test passes and you see that your changes are complete, as a developer, create a pull request o that your team knows that you trust the work on your branch is complete and that the code is ready to be examined for inclusion in the master branch.

10.2.2 Flow of a Build on a Master Branch

The following figure 10-2 shows the life cycle of a master branch build

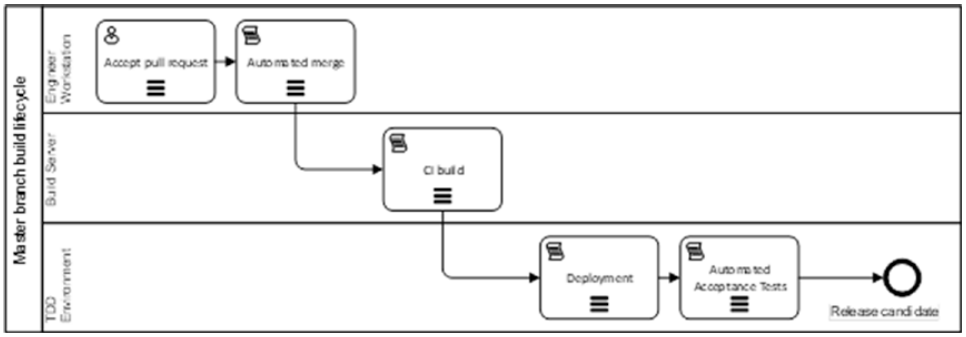


Figure 10-2. *The build process for changes on master end with a new release candidate*

Explanation:

- After the pull request is approved, your branch is automatically merged into master.

- This applies whether you are using GitHub or Azure Repos. The CI (Continuous Integration) build, which is monitoring for changes, will start off.
- If successful, the build artifacts will be stored in Azure Artifacts as NuGet packages. Then the build will be installed to the TDD environment for verification of deploy ability and for the running of the automated full-system acceptance tests.
- After successfully completing these acceptance tests, the builds considered a valid release candidate.
- In other words, it is a numbering candidate for a potential release, which can be further tested in a manual test environment (or even in other automated test environments) and deployed along the pipeline toward production.

10.2.3 Steps of a Build

- The private build runs on a developer workstation. The CI build runs on shared team build infrastructure, whether it is on a full server or in Azure Pipelines.
- Test-driven development (TDD) introduces the concept of Arrange, Act, and Assert.

- **The flow of Arrange, Act, and Assert is as follows:**

1. **Arrange:** In any verification, whether an automated testing, a manual testing, a static analysis run, or a CI build, the verification process is responsible for setting up an environment that can run.
2. **Act**: In this step, you will execute the process, run the some code, start a procedure, etc.
3. **Assert:** Finally you can see how it works. You check whether what you expected has happened. If what happens is as expected, your validation is successful. If it does not meet expectations, your validation will fail.

The following figure 10-3 shows the types of activities that are common in both private build and CI build.

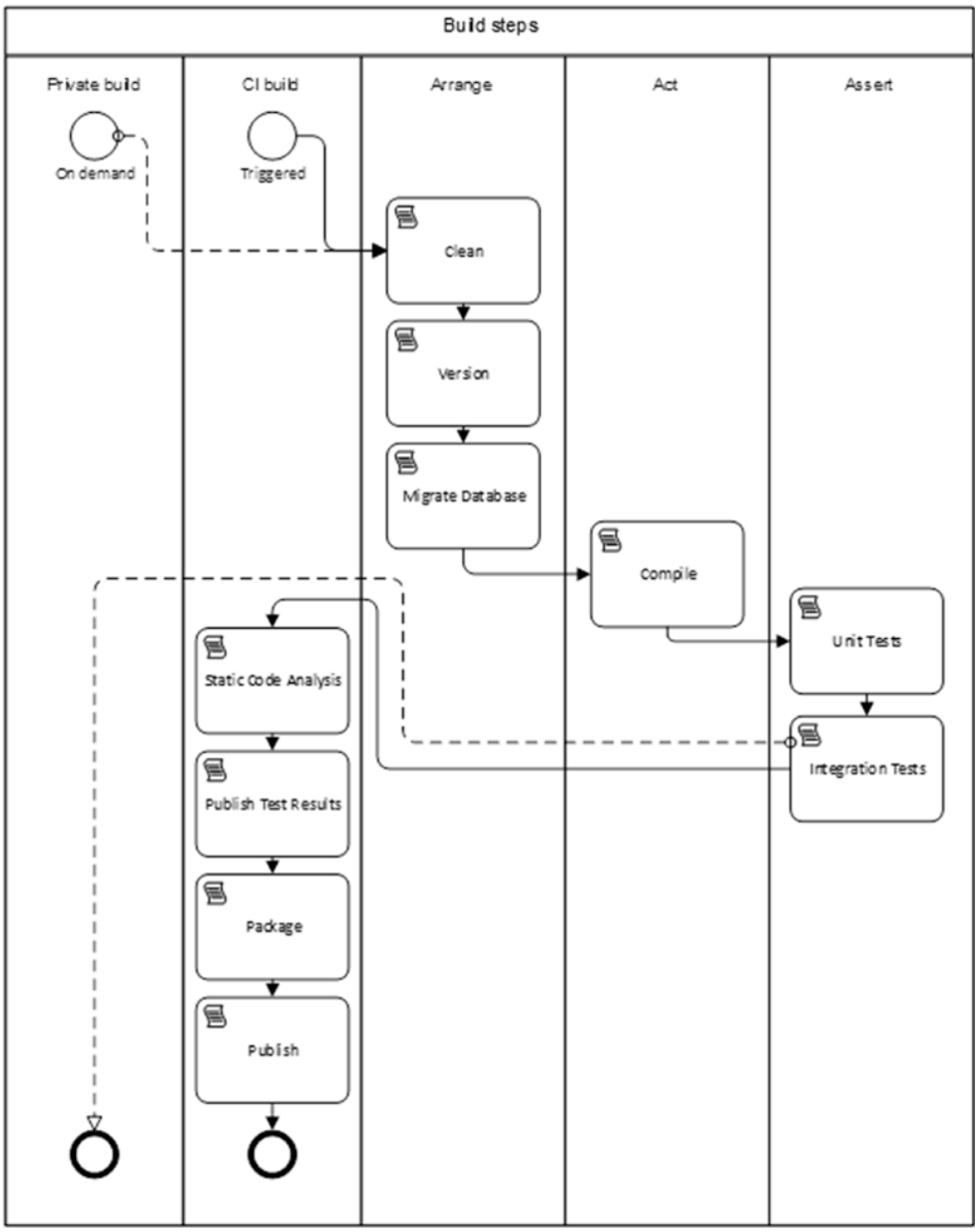


Figure 10-3. *The private and CI build have many steps in common*

Types of activities that are common in both private build and CI build:

- **Start:** The private build is triggered at the request of the developer. CI builds triggered by an observer on the Git repository- when a new commit occurs.
- **Clean:** Any temporary directories or files will be deleted, and the rest of the previous build will be deleted.
- **Version:** The build number is pushed into any areas of input needed for the resulting executable software to be labelled with the version number of the build. Private builds usually have a version coded as 0.0.0 or 9.9.9, so anyone looking at them can immediately tell that it is from a private build. In Azure Pipelines, the build number will come in from an environment variable, and the build script should push this

number into appropriate places, for example, an Assembly Info. cs file for .NET Framework or the dotnet.exe command line for .NET Core. If this step is skipped, then the resulting .NET assemblies will not be properly labelled with the build number.

- **Migrate Database:** This step represents everything environmental in which the application needs in order to function. Most applications store data, so you need to create a database and move it to the current schema to prepare for the subsequent build steps.
- **Compile:** This step converts the source files into assemblies, and performs any encoding, translation, reduction, etc., to convert the source code into a form suitable for execution in the intended runtime environment.
- **Unit Tests:** This is the first step that falls under the Assert category. It executes classes and functions that do not call out of process. In .NET, this is App Domain, which is a storage limit. Therefore, unit testing is very fast.
- **Integration Tests:** These tests ensure that the various components of the application can be integrated with each other. The most common is that our data access code can be incorporated into the SQL Server database architecture. The code run by these tests traverses the process (.NET App Domain, through the network stack to the SQL Server process) to test functionality. These tests are important, but they are orders of magnitude slower than unit tests. As the application grows, the ratio of unit testing to integration testing is expected to be approximately 10:1.
- **Private Build Success:** Perform a private build after these steps. Noneed to perform any other operations on the developer's workstation.
- **Static Code Analysis:** Static code analysis should be included in a CI build's list of validations. They're simple to run and can detect issues that automated tests can't.
- **Publish Test Results:** The CI build has completed successfully and the build artifacts must now be output. Each application type has a process that outputs the artifacts in a packaging-ready format.
- **Package:** This is the process of compressing each deployable application component, such as an ASP.NET web site, a database (SQL Server schema migration assets), a BatchJob (Windows service, Azure Function, etc.) and acceptance tests, into a named and versioned NuGet package. Azure Artifacts will receive these NuGet packages. Although zip files can be used, NuGet is the standard package format for.NET.
- **Publish:** This step involves uploading the packaged NuGet files to Azure Artifacts and making them available via the NuGet feed.
- **CI Build Success:** The continuous integration build is now complete, and we can report that it was successful.

10.3 USING BUILDS WITH .NET CORE AND AZURE PIPELINES

- Because of the compatibility and ease with which an automated continuous delivery pipeline can be set up with a software application located anywhere, Azure Pipelines is gaining popularity.
- Azure Pipelines may provide the build and deploy pipeline for GitHub or Azure Repos, as well as your own Git repository.
- **Continuous delivery has four steps. They are as follows:**
  1. **Commit:** The private build and continuous integration build are included in this stage.
  2. **Automated acceptance tests:** Your TDD environment, together with the test suites that represent acceptance tests, are included in the automated acceptance test stage.
  3. **Manual validations:** The UAT environment, is the deployed environment that can be used for manual validations.
  4. **Release:** In this final stage, your marketplace provides feedback on the value you created for them.

10.4 STRATEGY FOR DEFECT DETECTION

- Defect detection is one of the method which is used by software developers to evaluate software applications for bugs and defects.
- Because an application defect causes a difference between intended and actual results, it is important for a tester to track them down and fix them from the application.
- DRE (defect removal efficiency) is a metric that has been used in industry research for a long time.
- The three most important defect removal techniques are:
  1. Static analysis
  2. Testing
  3. Inspections

Below table 10-4 shows the average defect potentials by phase of work in a table.

Phase of work	Defects per 100 lines of C# (1 FP = ~52 C#)
Requirements	1.35
Architecture	0.19
Design	1.83
Code	2.21
Security code flaws	0.48
Documents	0.87
Bad fixes	1.25

Figure 10-4: Defects that should be expected by phase of work per 100 lines of resulting C# code

10.4.1. Strategy and Execution of Defect Detection

- There are many defect detections techniques. We will cover the three essential techniques. Let's take a look at each of the three main approaches for removing defects.
    1. Static Analysis
    2. Testing
    3. Inspections

10.4.1.1 Static Analysis

- The automatic evaluation of a source file in order to determine defects is known as static analysis. Static analysis is being used as a technique to documents and other artefacts as well as source code.
- Microsoft Word's spelling and grammar check is a very effective static analyser. The static analyser in Microsoft Word is run multiple times, often after every modification to the content,
- We'll use a variety of static analysis tools to examine our source code. As part of our DevOps pipeline, these will execute automatically.
- Warnings and errors will be generated by these tools. When mistakes arise, we can opt to fail a step in our pipeline, or we can choose to fail on new warnings.

10.4.1.2 Testing

- Testing is the process of assessing a system or its component(s) with the goal of determining whether or not it meets the specified requirements. Testing has always been an element of the software development process.
- A programmer has always tested the written code to ensure that it functions correctly.
- Test-driven development is a technique that allows developers to move away from manual desk checking and custom test harnesses and toward a standard pattern for building executable tests.
- The best approach for Scrum's acceptance criteria for a backlog item is writing test scenario whose steps are coded into an automated test that works the system

10.4.1.3 Inspections

- The most formal sort review is inspection, which is used during the static testing phase.
- A trained moderator, who is not the author, usually leads the inspection. The duty of the moderator is to conduct a peer review of a document.
- Entry and exit criteria are used in this review procedure. Before the meeting, the documents are prepared and checked thoroughly by the reviewers. It involves peer review of the product.
- A different preparation is done during which the product is evaluated and defects are detected. A logging list or issue log is used to keep track of the faults discovered.
- The moderator conducts a formal follow-up by applying exit criteria.

10.4.2 Code Validation in the DevOps Pipeline

- Work goes through our process in accordance with our swim lane.

Below figure 10-5 shows the standard swim lanes for a measurable DevOps process

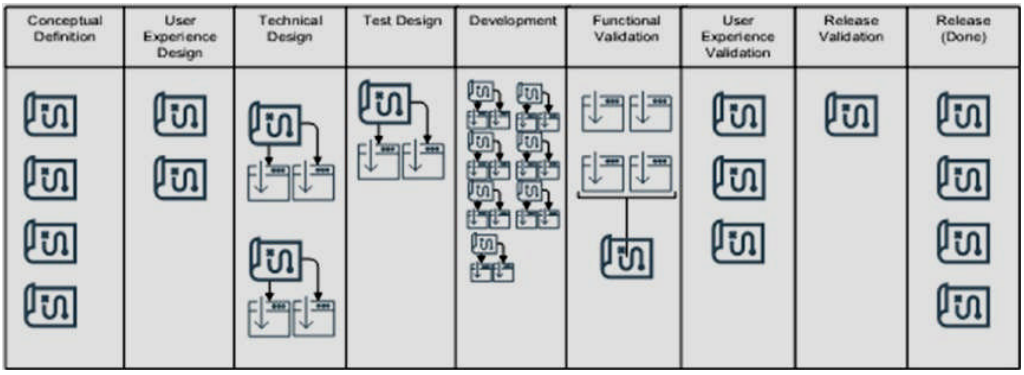


Figure 10-5 *Standard swim lanes for a measurable DevOps process*

- We will focus on just the following. These three stages of development enclose the code and result in a release candidate that may be tested further.

1. Test design
2. Development
3. Functional validation

Below figure 10-6 shows the part of our automated DevOps pipeline that will be impacted by the implementation of our defect removal methods.

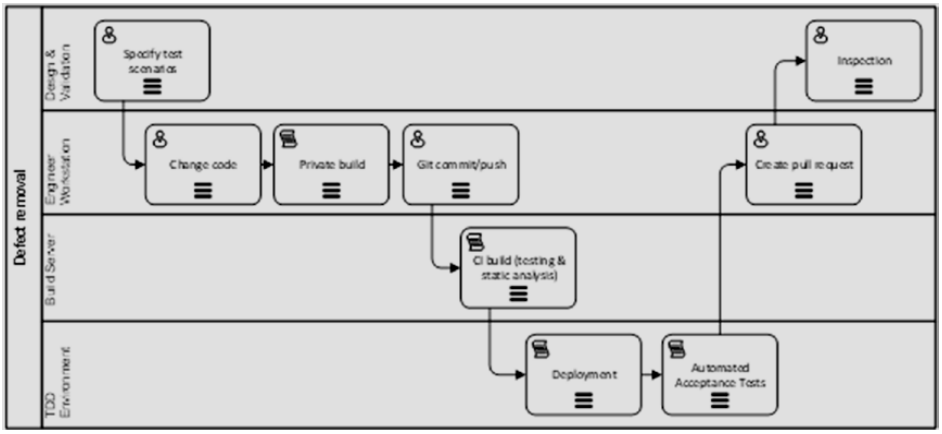


Figure 10-6: *validating the code starts a few steps before coding and includes some critical steps after*

10.4.3 Static Analysis

- In the DevOps development practice, it will occur in the create phases. Once the code is written, a static code analyser should be run to look over the code.

- You'll configure the static analysis tools in the continuous integration build once you've determined which ones to use.
- Any static analysis tool can be performed locally on demand, but you'll want to make it a part of your pipeline that runs automatically.
- It's important to put it before the release candidate packaging. FxCop has long been a.NET static analysis tool available in Visual Studio. Roslyn-based analysers have started replacing FxCop as a result of recent changes in the C# compiler and are now the recommended technique. These analysers are integrated into the Visual Studio solution and can run from both the IDE and the command line.
- There are many static analysis tools are available. **Some popular static analysis tools are:**
    1) **ReSharper Command Line:** For code style conventions
    2) **Ndepend:** For code metrics, warnings, and high-level quality grading
    3) **SonarQube:** For code metrics, warnings, and high-level quality grading
    4) **TSLint:** For readability, maintainability, and functionality errors
    5) **WAVE:** Web Accessibility Evaluation Tool for statically analyzing web pages for screen reader compatibility errors

10.4.4 Testing

- Mainly testing can be categorized as 1. Manual Testing 2. Automated Testing.
- For some validation, only a human eye can cover a defect that may affect customers. The majority of system functionality can be covered by forms of automated testing.
- We reduce the burden on manual testers by implementing stages of automated testing and ensure that persons conducting usability testing do not discover functional issues.
- Automated Testing can be categorised as

1) Unit tests
2) Integration tests
3) Full-system tests
4) Specialized tests

- Example, full-system tests are testing user scenarios with the fully deployed system online. Each branch of business logic can be tested as a unit test, and each branch of database or queue behaviour can be tested with an integration test.

10.4.4.1 Unit Tests (L0)

- These tests are really quick. The call stack is kept in memory at all times. The average execution time for these tests should be between 50 and 70 milliseconds.

- Code with out-of-process dependencies is excluded as a result of these.
- These tests can cover a single method or a group of classes, but they must cover a logical unit of programme logic.
- These tests should be able to execute on both the developer's computer and the build server.
- These tests should be included alongside the production code in the Visual Studio solution. Some anti-patterns for unit tests are
    1) Use of global or threading resources like Mutexes, file I/O, registry and so on
    2) Any dependencies between a test and another
    3) High consumption of CPU or memory for a single test
    4) Including code that calls out of the current process

10.4.4.2 Integration Tests (L1)

- Integration testing is used to ensure that modules/components perform as intended when they are combined, i.e. to ensure that modules that work well separately do not cause problems when combined.The best example of this is the database schema, the data layer, and the domain model entities.
- According to Microsoft, L1 tests should take less than 2 seconds to complete. The great majority of these tests should take less than a second to complete.
- These tests should be included in the Visual Studio solution with the production code. Some anti-patterns for integration tests are
    1) Requirement for large amounts of data setup.
    2) Any functional dependency on any other test.
    3) Validating more than one logical behavior between layers (being too large).
    4) Requiring external test state or data setup: Every test is responsible for its own setup.

10.4.4.3 Full-System Tests (L2)

- These tests are a subset of the specified test scenarios for each produced feature, as well as the defect fix proofs created once the root cause of a defect has been determined.
- To run full-system tests, you'll need a fully installed environment. E.g.-Selenium can be used to enter in text boxes and push buttons on a web page.
- These tests should run in the context of an identity and run the entire application just as a normal user. These tests should be included in the Visual Studio solution with the production code. Some anti-patterns for integration tests are

1) Unnecessarily Slow: While these tests will be a few orders of magnitude slower than unit tests, the aggregate of them will determine the cycle time of a release branch.
2) Modify global state.

3) The use of shared resources that prevent parallelization.
4) Requirement of third-party services that are outside of the team's control, that is, Office 365 login, PayPal, and the like.

10.4.5 Inspections

- Inspections are done by manually. However, it is not the same as manual testing. An inspection is a standardised procedure in which a human examines a work product using the same checklist and criteria as all other work products.
- The checklist for this type of inspection might have high-level items to verify completeness:
    1) Feature includes conceptual definition and vision description along with objectives.
    2) Feature includes detailed user experience design such as wire frames, screen mock-ups, and the like.
    3) Feature includes changes to architecture layers, new libraries needed, and other key technology decisions.
    4) Feature includes written test scenarios complete with test steps suitable for manual execution as well as test automation.
- A good implementation of an inspection would be connected with the pull request process for the purpose of determining coding defects. A pull request can manage and document the process of accepting changes on the branch back into the master branch if every feature/user stories is built on a feature branch.
- In Azure Repos or GitHub, the pull request experience is rich enough to accommodate a formal, documented inspection.
- Because the inspection's goal is source code, a power user or product owner would not be a qualified inspector for this type of inspection. But a product owner/product manager would likely be very interested in the results of the inspection, reports that they are happening, and the number of defects that are found and fixed through executing inspections.
- Along with other items, a pull request code inspection might have steps from the following list:
    1) The application works after a Git pull and private build.
    2) The changes conform to the approved architecture of system.
    3) The changes implement the design decisions called out in the feature.
    4) The changes conform to existing norms of the code base.
    5) No unapproved packages or libraries were introduced to the code base.
    6) The code is accompanied by right balance of tests.
    7) All test scenarios in acceptance criteria of the feature have been implemented as full-system L2 tests.
    8) Logging is implemented properly and of sufficient detail.

9) Performance Considerations: Application specific.

10) Security Considerations: Application specific and conforming to organizational standards.

11) Readability Considerations: Code is scannable – factored and named so that it is self-documenting and quickly reveals what it does.

10.5 IMPLEMENTING DEFECT DETECTION

- Let's look at how each of these defect detection technique looks in.NET and how to implement those using Azure DevOps Services

10.5.1 Static Analysis

- After adding FxCopanalyzers to a .NET Framework application, we can customize the built-in Microsoft rulesets right from within Visual Studio.

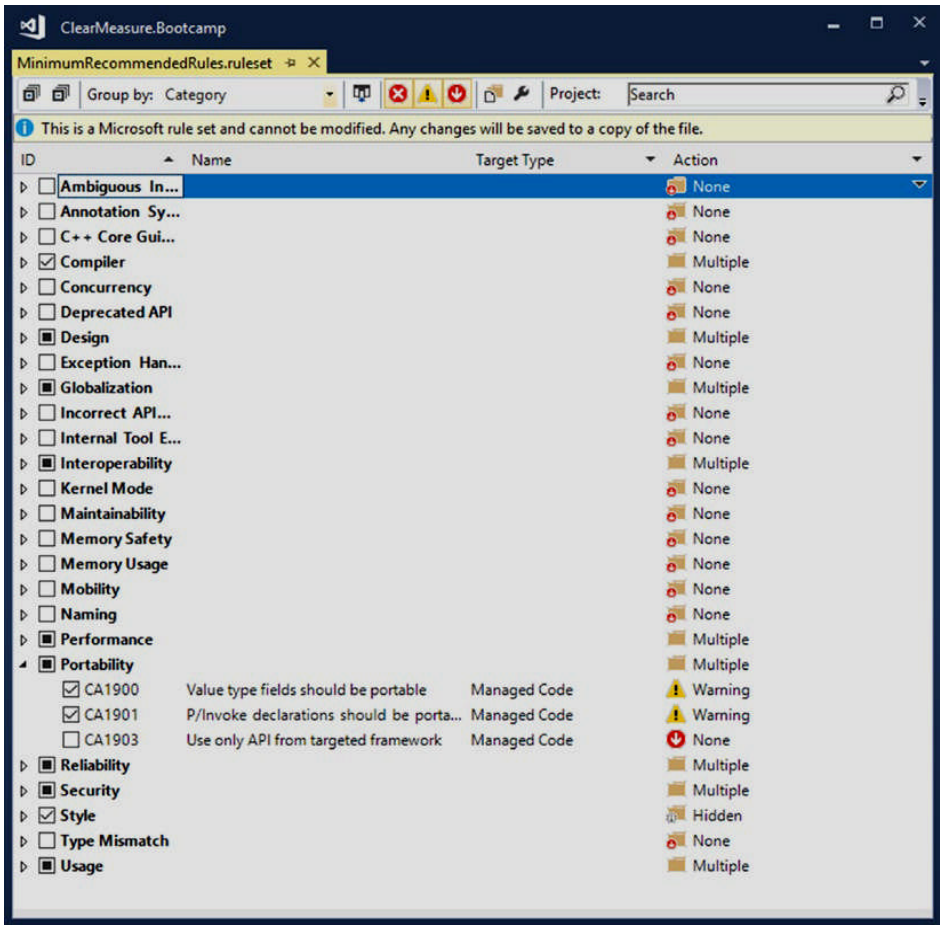


Figure10-7 *Visual Studio will save a project-specific ruleset file if you modify any of the settings of the Microsoft ruleset*

- In your build script, you can add the following command-line arguments so that the analysers are run when you want them run. Make sure to fail the build on a rule failure:

```
msbuild.exe
```

```
/t:Clean`;Rebuild /v:m /maxcpucount:1 /nologo
/p:RunCodeAnalysis=true
/p:ActiveRulesets=MinimumRecommendedRules.ruleset
/p:Configuration=Release
src\MySolution.sln
```

- When you add the NuGet package “Microsoft. Code Analysis. FxCop Analyzers” to your project in .NET Core, you’ll see the analyzers appear in your Solution Explorer, and warnings will start to show when you build your code inside Visual Studio, as shown in Figure 10-8

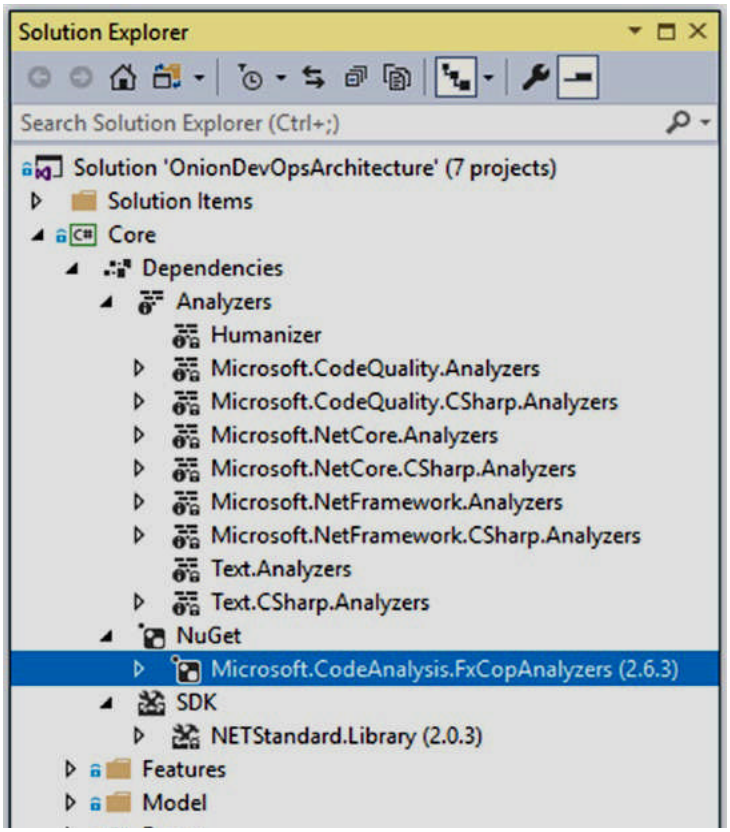


Figure 10-8 *Code analyzers are added to a .NET Core project through NuGet*

- There's no need to include a command-line arguments to your build script's call to dotnet.exe. When you add analyzers to your project, they will run automatically and generate the necessary warnings or errors.
- Each static analysis product has its own set of instructions for integrating it with your code, but to keep your Azure Pipelines build settings simple, include your static analysis tools in your build script so that the configuration is saved in your Git repository.
- You'll be keeping additional build logic in Git if you convert your Azure Pipelines build to YAML.

10.5.2 Testing

More specifically we will see following:

1) Unit Tests
2) Integration Tests
3) Full-System Tests

10.5.2.1 Unit Tests

- In our example application, we have an entity which serves as an aggregate root, in domain-driven design terms. It has a number of properties and methods. The code for this short class is as follows:

```
using System;
namespace Clear Measure. Onion DevOps Architecture. Core.Model
{
public class ExpenseReport
{
public Guid Id { get; set; }
public string Title { get; set; }
public string Description { get; set; }
public ExpenseReportStatus Status { get; set; }
public string Number { get; set; }
public ExpenseReport()
{
Status = ExpenseReportStatus.Draft;
Description = "";
Title = "";
}
public string FriendlyStatus
{
get { returnGetTextForStatus(); }
}
protected string GetTextForStatus()
{
return Status.ToString();
}
public override string ToString()
{
return "ExpenseReport " + Number;
}
protected bool Equals(ExpenseReport other)
{
return Id.Equals(other.Id);
}
```

```
public override bool Equals(object obj)
{
if (ReferenceEquals(null, obj)) return false;
if (ReferenceEquals(this, obj)) return true;
if (obj.GetType() != this.GetType()) return false;
return Equals((ExpenseReport) obj);
}
public override int GetHashCode()
{
return Id.GetHashCode();
}
}
}
```

- There's a lot of logic here that may go wrong. We can build some unit tests because this functionality can be tested within a single memory space without having to call out of process to any application dependencies.
- Some methods are used by the base class library (BCL) in a code base where entities are placed into collections, sorted, and compared, and exhibit a reduced return on investment for explicit unit testing. These methods are Equals() and Get Hash Code().
- Any entity in a domain model that does not implement these will force additional logic to figure out which attribute represents its identity in order to determine whether two objects represent the same record.
- Most of these objects have data that is pulled from a database of some sort. Full coverage on Equals() and Get Hash Code() normally happens automatically as tests of business logic are written.
- And some tools such as Jet Brains Re Sharper will generate these methods automatically, so the likelihood of defects is low unless you handwrite them.

A unit test class for Expense Report is shown here:

```
using System;
using ClearMeasure.OnionDevOpsArchitecture.Core.Model;
using NUnit.Framework;
namespace ClearMeasure.OnionDevOpsArchitecture.UnitTests
{
public class ExpenseReportTester
{
[Test]
public void PropertiesShouldInitializeToProperDefaults()
{
var report = new ExpenseReport();
```

```
Assert.That(report.Id, Is.EqualTo(Guid.Empty));
Assert.That(report.Title, Is.EqualTo(string.Empty));
Assert.That(report.Description, Is.EqualTo(string.Empty));
Assert.That(report.Status, Is.EqualTo(ExpenseReportStatus.
Draft));
Assert.That(report.Number, Is.EqualTo(null));
}
[Test]
public void ToStringShouldReturnNumber()
{
var report = new ExpenseReport();
report.Number = "456";
Assert.That(report.ToString(), Is.EqualTo("ExpenseReport 456"));
}
[Test]
public void PropertiesShouldGetAndSetValuesProperly()
{
var report = new ExpenseReport();
Guidguid = Guid.NewGuid();
report.Id = guid;
report.Title = "Title";
report.Description = "Description";
report.Status = ExpenseReportStatus.Approved;
report.Number = "Number";
Assert.That(report.Id, Is.EqualTo(guid));
Assert.That(report.Title, Is.EqualTo("Title"));
Assert.That(report.Description, Is.EqualTo("Description"));
Assert.That(report.Status,
Is.EqualTo(ExpenseReportStatus.Approved));
Assert.That(report.Number, Is.EqualTo("Number"));
}
[Test]
public void ShouldShowFriendlyStatusValuesAsStrings()
{
var report = new ExpenseReport();
report.Status = ExpenseReportStatus.Submitted;
Assert.That(report.FriendlyStatus, Is.EqualTo("Submitted"));
}
}
}
```

- As you read through this code file, you'll see that each test verifies that a piece of logic functions correctly while keeping the rest of the code running. Unit tests built in this manner run instantly, with thousands of them running in seconds.

10.5.2.2 Integration Tests

- Our Expense Report object is persisted, through Entity Framework Core, to a SQL Server database. In order to validate that the expense report class can be hydrated from data inSQL Server, we need a test that puts several layers together:

1) The domain model itself, containing the expense report class
2) The Entity Framework Core mapping configuration
3) The data access logic, specifying the query to run
4) The SQL Server schema, which contains the DDL (data definition language) for the Expense Report table

- These tests are usually simple to construct, but they are extremely critical. You will discover defects if you don't have them, and you will waste time debugging through these four layers to find the problem.
- If all of your database-backed classesare equipped with persistence-level integration tests, you will seldom find yourself in adebugging session for a problem in this area.
- We have seen the expense report class. The next class to examine is the Entity Framework Core mapping configuration, which is comprised of the data context class and a mapping class. The data context class is as follows:

```
using ClearMeasure.OnionDevOpsArchitecture.Core;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Diagnostics;
namespace
ClearMeasure.OnionDevOpsArchitecture.DataAccess.Mappings
{
public class DataContext :DbContext
{
private readonlyIDataConfiguration _config;
public DataContext(IDataConfiguration config)
{
_config = config;
}
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
optionsBuilder.EnableSensitiveDataLogging();
var connectionString = _config.GetConnectionString();
```

```
optionsBuilder
.UseSqlServer(connectionString)
.ConfigureWarnings(warnings =>
warnings.
Throw(RelationalEventId.QueryClientEvaluationWarning));
base.OnConfiguring(optionsBuilder);
}
protected override void OnModelCreating(ModelBuildermodelBuilder)
{
new ExpenseReportMap().Map(modelBuilder);
}
}
}
```

- In our example application, we have one aggregate root, so in our On Model Creating class, we include one “Map” class. We use this pattern so that as we accumulate hundreds mapped entities, each has its own class rather than bloating the single Data Context class:

```
using System;
using ClearMeasure.OnionDevOpsArchitecture.Core.Model;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore.ValueGeneration;
namespace
ClearMeasure.OnionDevOpsArchitecture.DataAccess.Mappings
{
public class ExpenseReportMap :IEntityFrameworkMapping
{
public EntityTypeBuilderMap(ModelBuildermodelBuilder)
{
var mapping = modelBuilder.Entity<ExpenseReport>();
mapping.UsePropertyAccessMode(PropertyAccessMode.Field);
mapping.HasKey(x =>x.Id);
mapping.Property(x =>x.Id).IsRequired()
.HasValueGenerator<SequentialGuidValueGenerator>()
.ValueGeneratedOnAdd()
.HasDefaultValue(Guid.Empty);
mapping.Property(x =>x.Number).IsRequired().HasMaxLength(10);
mapping.Property(x =>x.Title).HasMaxLength(200);
mapping.Property(x =>x.Description).HasMaxLength(4000);
mapping.Property(x =>x.Status).HasMaxLength(3)
```

```
.HasConversion(status =>status.Code
, s =>ExpenseReportStatus.FromCode(s));
return mapping;
}
}
}
```

- Rather than rely on defaults, which tend to change, our map class specifies howto map each property. Choosing to be explicit in this fashion also lowers the bar fordevelopers understanding what is going on. Each developer will have a different level ofmemorization for what Entity Framework Core's default behavior is.
- Our ExpenseReport table looks like the following:

```
CREATE TABLE [dbo].[ExpenseReport] (
[Id] UNIQUEIDENTIFIER NOT NULL,
[Number] NVARCHAR (10) NOT NULL,
[Title] NVARCHAR (200) NULL,
[Description] NVARCHAR (4000) NULL,
[Status] NCHAR (3) NOT NULL
);
```

- With four different layers of code running across two different processes, most of the time across a network on different servers, you should see the importance of an automated test ensuring the stability of the integration of these layers. Our integration test to validate persistence logic is here:

```
using ClearMeasure.OnionDevOpsArchitecture.Core.Model;
using NUnit.Framework;
using Shouldly;
namespace
ClearMeasure.OnionDevOpsArchitecture.IntegrationTests.DataAccess.
Mappings
{
public class ExpenseReportMappingTester
{
[Test]
public void ShouldPersist()
{
new DatabaseTester().Clean();
var report = new ExpenseReport
{
Title = "TestExpense",
Description = "This is an expense",
```

```
Number = "123",
Status = ExpenseReportStatus.Cancelled
};

using (var context = new StubbedDataContextFactory().
GetContext())
{
context.Add(report);
context.SaveChanges();
}
Expense Report rehydrated Expense Report;
using (var context = new Stubbed Data Context Factory().
Get Context())
{
rehydrated Expense Report = context
.Find<Expense Report>(report.Id);
}
rehydrated Expense Report.Title.Should Be (report.Title);
rehydrated Expense Report. Description. ShouldBe(report.
Description);
rehydratedExpenseReport.Number.ShouldBe(report.Number);
rehydratedExpenseReport.Status.ShouldBe(report.Status);
}
}
}
```

- This integration test technique can be applied to any class that has to be persisted to a database using an object-relational mapper. The most basic scenario is to send an object to the database via the ORM, clear memory, and then query again to populate the object.
- The call to Database Tester. Clean() represents a helper that can remove all records from all tables in the database in the order of foreign key dependencies.
- In integration tests involving a database, each test is responsible for putting the database in a known state. In many cases, it can be appropriate to run a test starting with no records in the database.

10.5.2.3 Full-System Tests

- Full-system tests and acceptance criteria implementation should start at the application's external interfaces.
- If the feature in question is a web service, the test should call the web service and execute setup. The test should navigate to and use the user interface screen if the interface is a user interface screen. The test should create an Excel file and set it in the correct file directory to be

processed if the interface is file ingestion of a bespoke Excel file for data import.

- For a simple form-based login screen, a Selenium test might look similar to the following:

```
[Test]
public void ShouldLoginAndLogOut()
{
Driver.Navigate().GoToUrl(AppUrl);
var login = Driver.FindElement(
By.XPath("//button[contains(text(), 'Log In')]"));
login.Click();
Driver.Title.ShouldStartWith("Home Page");
var logout = Driver.FindElement(By.LinkText("Logout"));
logout.Click();
Driver.Title.ShouldStartWith("Login");
}
```

- Driver is a property in this case that refers to the Selenium Driver class, which wraps a model of the web page being seen by the browser. These tests can run on any machine where the executing identity has the ability to initiate and control a web browser instance.
- Because full-system tests must be conducted against a fully deployed environment, the CI build process must package the test suite and deploy it alongside the application components in the TDD environment.

10.5.3 Inspections

- A pull request in Azure Repos or GitHub is ideal for facilitating a code review.
- Here's an example of a flow in Azure Repos. We begin with a feature branch that is ready to be merged. The pull request is created by the developer.
- The developer must start the description with a markdown task list that contains all of the inspection procedures. This can be retrieved from a wiki or a markdown file stored on your computer.

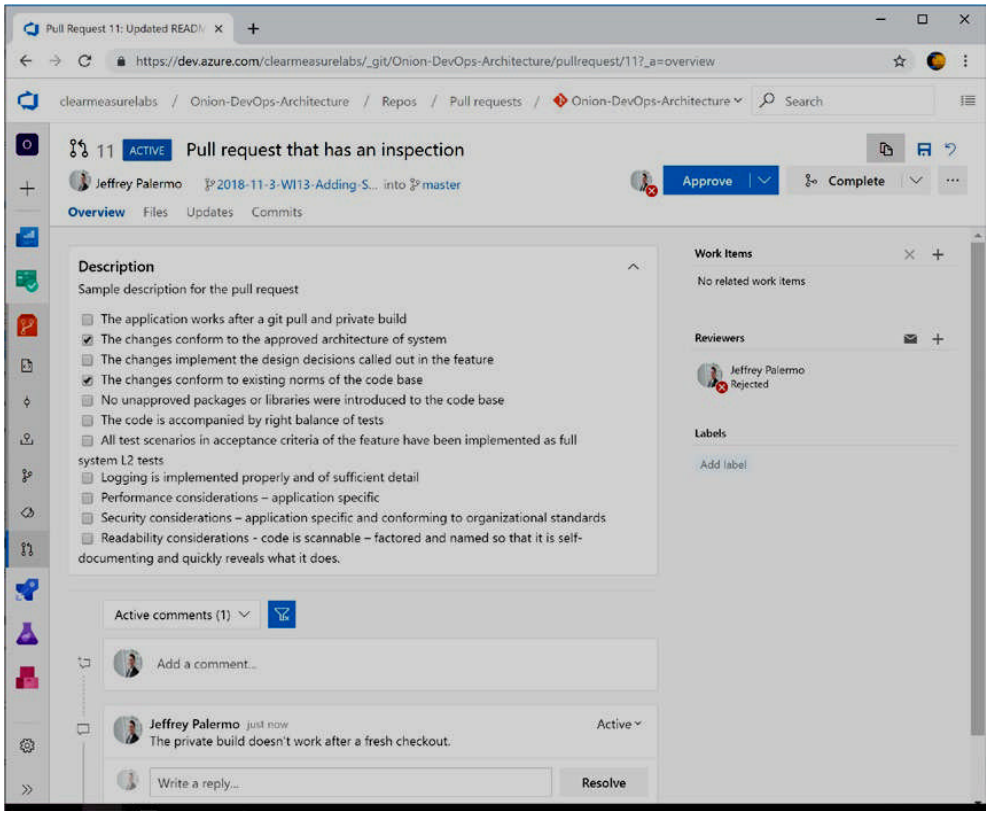


Figure10-9 *Pull request that executes a multistep inspection*

- As the items are inspected, the approver can check them off. When a task fails, the comments can be used to deny the pull request.
- To fix the problem, more commits might be added to the branch. The submitter can then request that the inspector take another look using the comments in the pull request.
- The inspector authorises the pull request and merges the branch once it fulfils all of the inspection criteria.
- In Azure Repos, the checklist, as well as the entire dialogue required to fix any errors, is completely documented.

SUMMARY

In this chapter you learnt how to structure your code. You've learnt how to set up each type, as well as the structure of a build. You've seen how a build flow differs between a feature branch and a master branch. You've also learned how to develop a.NET Core solution using Azure Pipelines. You've also learned three of the industry's most effective critical defect removal techniques. Static analysis, several levels of testing, and the notion and execution of inspections have all been discussed.

REVIEW QUESTIONS

THEORY QESTIONS

1. Explain the structure of build.
2. Explain flow of a build on feature branch.
3. Explain the flow of build on master branch.

   OR

   Explain life cycle of a master branch build.
4. Explain the various steps of build.

   OR

   Explain the types of activities that are common in both private and CI build
5. Explain 4 steps of continuous delivery.
6. Explain 3 defect removal techniques.

   OR

   Explain any one defect removal technique
7. Explain standard swim lanes for a measurable DevOps process.
8. Explain Categories of automated testing.
9. Explain implementation defect detection of static analysis, testing and inspection (Any one).

MULTIPLE CHOICE QESTIONS

1. _____ build runs only on the developer's workstation.
a) Public b) Integration c) Private d) Protected

2. _____ build runs only on the shared server.
a) Public b) Integration c) Private d) Protected

3. After the pull request is approved, your branch is automatically merged into _____.
a) Slave b) Centre c) Master d) User

4. Full form of CI is _____.
a) Continuous Integrated b) Continuous Integrating
c) Continuous Integrate d) Continuous Integration

5. _____ build runs on shared build infrastructure
a) Private b) CI c) Integration d) Public

6. _____ step will execute the process, run the same code, start a procedure etc.

a) Act b) Arrange c) Assert d) Action

7. _____ step converts the source file into assemblies and performs any encoding, translation, reduction etc.

a) Start b) Unit Test c) Migrate Database d) Compile

8. _____ tests ensure that the various components of the application can be integrated with each other.

a) Integration b) Unit c) Acceptance d)User Acceptance

9. In _____ stage, the private build and continuous integration build are included.

a) Release b) Commit c) Compile d) Start

10. In _____ stage, your marketplace provides feedback on the value you created for them.

a) Release b) Commit c) Compile d) Start

11. Full form of DRE is _____.

a) Debug Removal Efficiency
b) Defect Removal Efficiency
c) Debug Removal Efficient
d) Defect Removal Efficient

12. Which of the following is not a defect removal technique?

a) Testing b) Execution c) Static Analysis d) Inspection

13. The automatic evaluation of source file in order to determine defects is known as _____.

a) Dynamic Analysis b) Parallel Analysis
c) Static Analysis d) Serial Analysis

14. _____ is the process of accessing a system or its components with goal of determining whether or not it meets the specified requirements.

a) Execution b) Production c) Coding d) Testing

15. _____ tests are testing user scenarios with the fully deployed system online.

a) Unit b) Integration c) Full System d) Specialized

16. The average execution time of unit tests should be between ___ And ____ milliseconds.

a) 50,60 b) 40,70 c) 50,70 d) 40,50

17. _____ tests should be able to execute on both the developer's computer and the build server.

a) Integration b) Unit c) Full System d) Specialized

18. _____ testing is to ensure that modules/components perform as intended when they are combined.

a) Unit b) Full System c) Integration d) Specialized

19. According to Microsoft, L1 test should take less than ___ second/seconds to complete.

a) 1 b) 3 c) 5 d) 2

20. _____ tests are subset of specified test scenarios for each product feature.

a) Unit b) Full System c) Integration d) Specialized

21. _____ tests should run in the context of an identity and run the entire application just as the normal user.

a) Unit b) Full System c) Integration d) Specialized

22. Inspections are done ____

a) Manually b) Automatically c) Serially d) Locally

23. _____ is a standardised procedure in which human examines a work product using the same checklist and criteria as all other work products.

a) Execution b) Inspection c) Coding d) Production

REFERENCES

- Duvall, P. M. (2007). Continuous Integration: Improving Software Quality and Reducing Risk. Addison Wesley.
- Jeffrey Palermo. .NET DevOps for Azure a Developer's Guide to DevOps Architecture the Right Way, Apress (2019)

❖❖❖❖

11

RELEASE CANDIDATE CREATION

Unit Structure :

11.0 OBJECTIVE

In this chapter you will learn the following concepts:

- How to create and use release candidate packages
- Specifying how packages are created
- How to use release candidate packages in deployment

11.1 INTRODUCTION

Internally, a Release Candidate (RC) build is released to see if any serious bugs were introduced into the code during the previous development phase. Release candidates are strictly for testing reasons and should not be used in production. This chapter focuses on the key elements you'll need to create and configure to turn a build into a versioned release candidate that can be deployed to downstream environments. This chapter will also cover the principles involved, the model and relationships of packages to the software architecture, and the process for storing and using packages.

11.2 DESIGNING YOUR RELEASE CANDIDATE ARCHITECTURE

You must examine the logical and physical layers of your 4+1 architecture (The 4+1 view model is used to "describe the structure of

software-intensive systems using several, concurrent views.")and identify the unit of deployment in order to establish the architecture for your release candidate packages.

There are the following rules of thumb for release candidate packages:

1. **Build/package once, deploy many:**

- If the continuous integration build succeeds, a set of release candidate packages will be packaged. These packages should be able to be deployed in any environment, including production.
- Do not configure branches or builds in such a way that each environment is built independently.
- We compile and package once so that every subsequent action in our DevOps pipeline verifies the release candidate's suitability for release to our production environment users.
- When a defect is identified, the release candidate loses its ability to be released.
- We can be confident that a release candidate has passed through our entire pipeline and is ready for our users.

2. **One package per runtime component:**

- A release candidate is typically made up of a collection of packages (For example, NuGet packages for our.NET apps).
- A package is made up of each runtime component in our physical architectural layer. That is, our complete website is contained within a NuGet package.
- Our SQL Server database schema and migration scripts, for example, are packaged separately as a NuGet package.

3. **Use the NuGet package format:**

- While anyone can create a new package format, it's best to go with an industry standard that includes tools like viewers.
- A.*.nupkg file is the.NET package format (pronounced NUPKEG).
- While you could technically use a.zip file to store your files, you're better suited choosing a platform-specific format.
- Similarly, you would use the.*.npm package format if you were developing a NodeJS application.

4. **Embed the build number as the release candidate version:**

- Build numbers are assigned automatically based on the format you specify.
- Ensure that the build number is applied to each package in your release candidate.
- If you're using build 3.4.352, all of the release candidate's NuGet packages should be labelled 3.4.352.
- This ensures that you always know which release candidate is being tested and which build created it, regardless of the environment.

5. **Package only application artifacts:**

- Make sure you don't include any environment-specific files or configuration in your package.
- Each application component package should contain content that is suitable for deployment in any environment.
- Global configuration is appropriate, but any configuration that varies by environment should be applied at deploy time instead of package time.

11.2.1 Creating and Using Release Candidate Packages

The following figure 11-1 shows the sequence of events in our DevOps Pipeline:

Explanation:

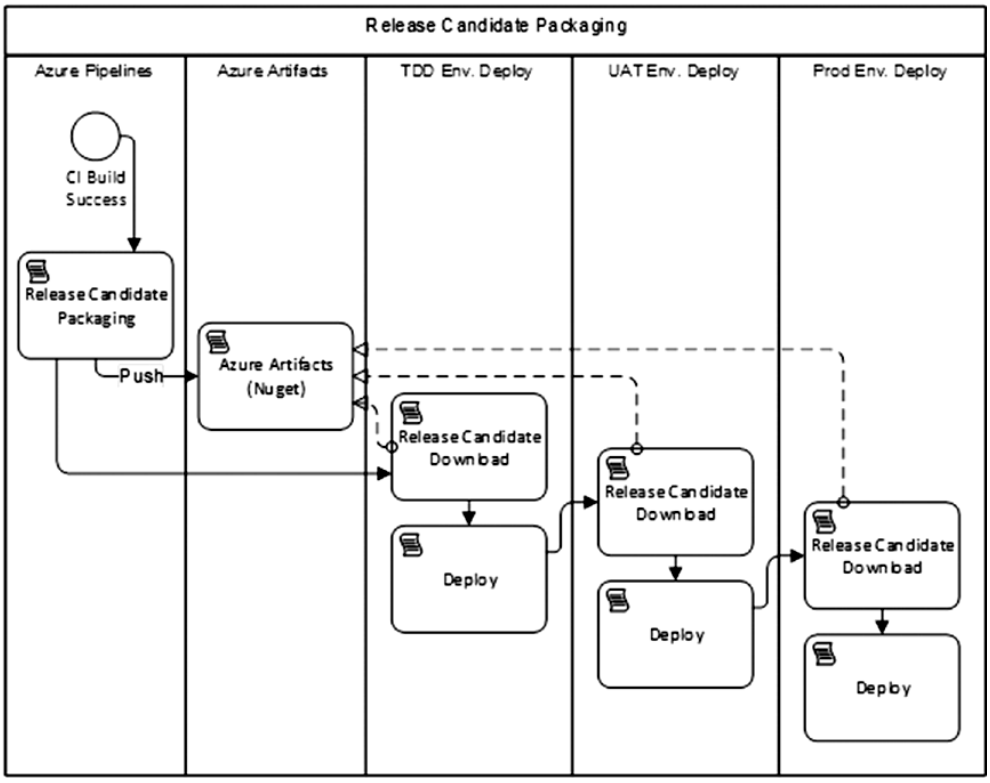


Figure 11-1: Release candidate packages are the bridge between a build and deployment

- The CI build generates release candidate packages, which are then used by the deployment configuration.
- Regardless of the number of runtime components in your application, Azure Pipelines will provide you with a single continuous integration build.
- You'll only have one build for a single Git repository containing a Visual Studio solution. Upon completion of all stages, that build

should package your application for deployment as a group of NuGet packages.

- The build configuration should push the release candidate into Azure Artifacts once it has been packaged.
- For your team, Azure Artifacts has a built-in NuGet package.
- The CI build finishes and reports success after the release candidate packages have been pushed. The build should fail if the release candidate cannot be packaged or stored.
- The deployment process will begin with the retrieval of the NuGet packages for the release candidate in each of the downstream environments (TDD, UAT, and Prod).
- This process will retrieve the packages from the NuGet feed hosted in Azure Artifacts, extract them, and use the contents to deploy your application.
- For deployment to each environment, the same packages that make up a single release candidate should be used.

11.2.2 Defining the Bounds of a Package

There are multiple runtime components in most applications. As a result, there should be more than one NuGet package in the set of packages that make up a release candidate.

The following figure 11-2 shows that each part of your application that is deployed a specific way should be packaged separately

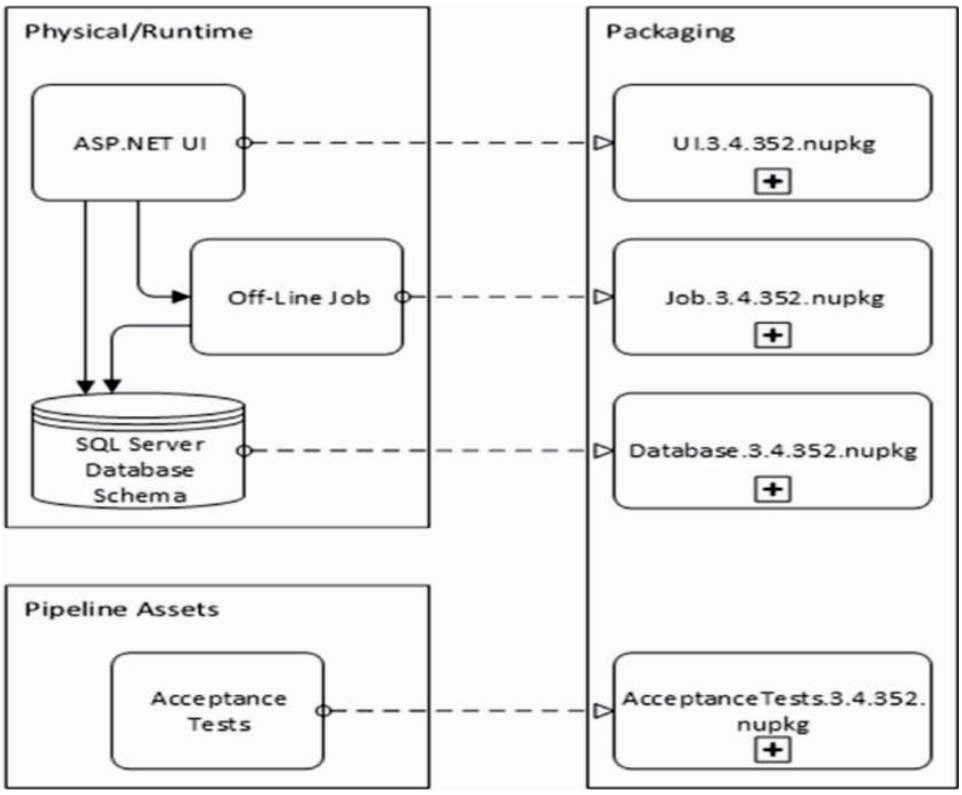


Figure 11-2: Each runtime component of the application should have its own package

Explanation:

- **Consider the following scenario:** an application with three deployable components.

1. The first component is an **ASP.NET web application**. It can be installed on a web server or in Azure App Service.
2. The second component is an **off-line job**. A batch job or a handler service listening to a queue, for example. These are common, and they're typically deployed as Windows services, Azure Functions, or WebJobs.
3. A **SQL Server database** is the third component. We must deploy schema and global data modifications whether on-premises or in Azure SQL.

- Because each of these three application components runs in its own memory area and in its own process, they have different deployment characteristics.
- The deployment location is not the same. As a result, each should be packaged separately in a NuGet package.
- This allows each to be delivered to the most appropriate location while maintaining flexibility.
- While all three components may technically be installed on the same server, they could also be installed on different servers.
- In fact, the web application ASP.NET UI might be deployed over several servers in a web farm or across multiple Azure regions.
- The structure of your release candidate packages should correspond to the physical architecture of the application's running components, disregarding the server environment's topology.
- You will always have some additional assets that go along with the release candidate in addition to each application component. These assets aren't meant to be used in production; they're only there to verify the release candidate.
- “Acceptance Tests” are illustrated in above Figure11-2. This is represented in Visual Studio as a project, most likely NUnit or another testing framework.
- If the application is a web application, the Visual Studio project's tests will always use Selenium to test the entire deployed application. Because a fully deployed application is required, these tests are attached to the application's version; as a result, they belong to a specific release candidate and must be packaged and deployed with the other components.
- The acceptance test package will be downloaded and installed on a deployment server when the application has been properly deployed.

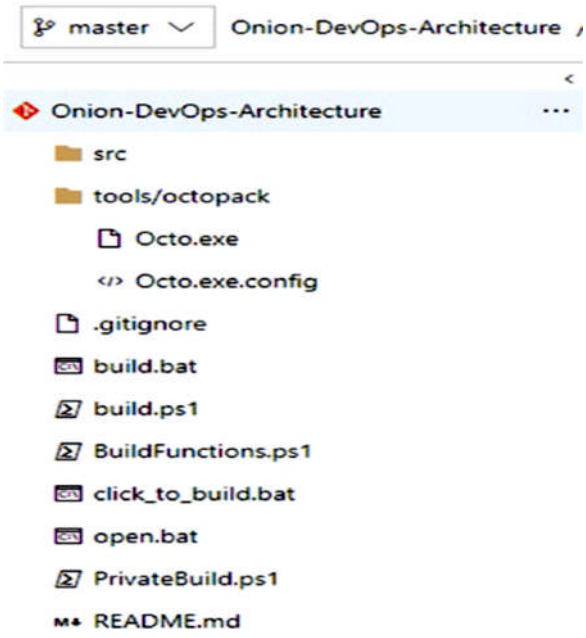
- The tests are then executed against the deployed release candidate in the TDD environment.
- In this way, we can packaged more assets to improve the robustness of our DevOps pipeline, allowing it to detect a higher percentage of errors before moving the release candidate to the next downstream environment.

11.3 AZURE ARTIFACTS WORKFLOW FOR RELEASE CANDIDATES

- You may develop and share npm, NuGet package feeds from public and private sources with teams of any size using Azure Artifacts.
- With a single click, you can integrate fully integrated package management into your continuous integration/continuous delivery (CI/CD) pipelines.
- It is an extension to Azure DevOps Services and Azure DevOps Server
- Azure Artifacts is a stand-alone product that works in conjunction with Azure Pipelines.
- It's the service that stores the release candidate components that the continuous integration build generates.
- We'll be packaging an application with **three deployable components** that are all developed and versioned together:
  i. Web site user interface (UI)
  ii. Off-line job
  iii. Database
- This application also contains acceptance tests that will be packaged and deployed in addition to these application components.
- We must package and save the version of the acceptance tests that belong to this version of the application in order to execute them against our application in the TDD environment. There must be a match between the version numbers.
- Your code gets access to the right version number because it is now encoded in every assembly.
- You'll use the version number to determine whether a defect or bug existed in a previous or current version. You're flying blind without the version number.

11.3.1 Specifying How Packages are created

The following figure 11-3 shows our Power Shell build script is stored at the top of the Git repository and is named build.ps1



- Packaging is not required when executing our private build on a local workstation, but when this script is run as part of a CI build, packaging is the last step before the build should succeed.
- This PowerShell function is responsible for creating NuGet packages for the projects.

```
Function Pack{
Write-Output "Packaging nuget packages"
exec{
&dotnet publish $uiProjectPath -nologo --no-restore --no-build
-v $verbosity --configuration $projectConfig
}
exec{
& .\tools\octopack\Octo.exe pack --id "$projectName.UI"
--version $version
--basePath $uiProjectPath\bin\$projectConfig\$framework\publish
--outFolder $build_dir --overwrite
}
exec{
& .\tools\octopack\Octo.exe pack --id "$projectName.Database"
--version $version --basePath $databaseProjectPath
--outFolder $build_dir --overwrite
}
exec{
&dotnet publish $jobProjectPath -nologo --no-restore --no-build
-v $verbosity --configuration $projectConfig
}
exec{
& .\tools\octopack\Octo.exe pack --id "$projectName.Job"
--version $version
--basePath $jobProjectPath\bin\$projectConfig\$framework\publish
--outFolder $build_dir --overwrite
}
```

- We have four NuGet packages because we have four components to deploy. I'm using the Octo.exe tool, as you can see.
- The full name is OctoPack, and it's an open source wrapper for NuGet that you can discover on GitHub.
- Before being adopted for application packaging, Nu Get was created as a package format for library dependencies.
- We may configure Azure Pipelines using the preceding Power Shell in our build script, as illustrated in following figure 11-4:

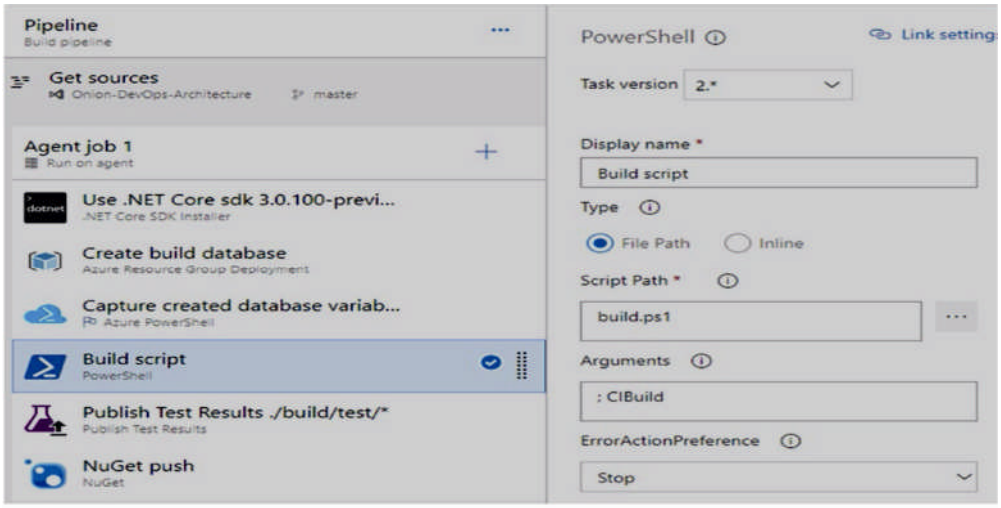


Figure 11-4 Azure Pipelines calls the build script stored in Git in order to minimize the global step configuration

- In above figure 11-4 notice that in the build.ps1 file, you'll notice that the "CIBuild" function is used. The following is the function:

```
Function CIBuild {
        Init
        MigrateDatabaseRemote
        Compile
        UnitTests
        IntegrationTest
        Pack
}
```

- The last function to be called is Pack. After that, our "NuGet push" build step places the NuGet packages in the Azure Artifacts services, making them available to any other process that accesses the NuGet feed.
- We can perform the database migration process from whatever server we want because we have the database migration tool and the full set of scripts.

11.3.2 Use Release Candidate Packages in Deployment

The following Figure 11-5 shows how a build or release configuration can obtain a NuGet package from Azure Artifacts.

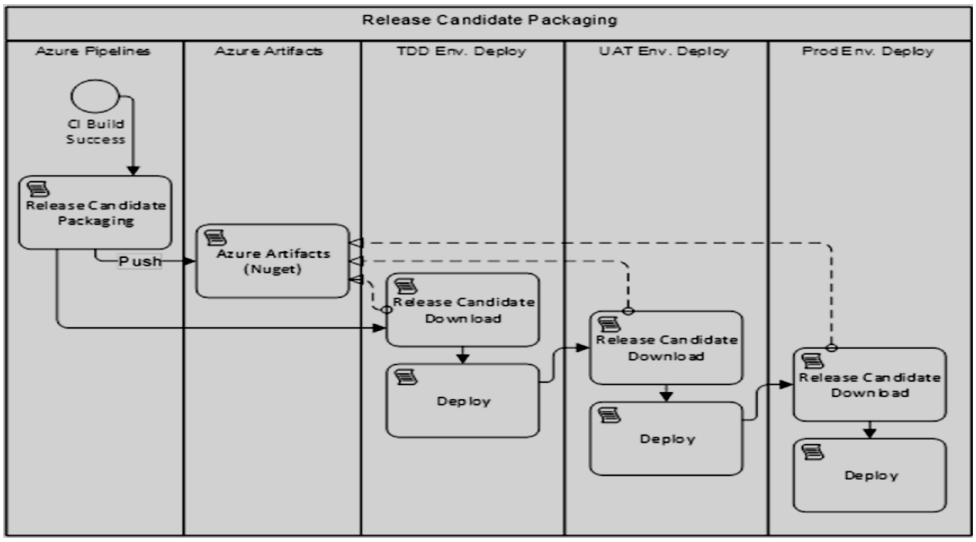


Figure 11-5 Review of how environment deployments call out to Azure Artifacts to obtain packages

Explanation:

- If you're using TDD, you'll need to deploy the entire application first, then pull and install the Acceptance Tests package to run the tests.
- Acceptance Tests is only found in the TDD environment. If you need to run other forms of test suites in a fully deployed environment, you'll follow the same approach of packaging them as a package, storing them in Azure Artifacts, and retrieving them during deployment.
- The "Download Package" step makes it simple to get our set of NuGet packages for any release candidate.
- This step fetches and extracts the NuGet package from the destination directory of the server that is used as the agent, whether it is a hosted agent or your own private agent, because a NuGet package is essentially a.zip file with a manifest.

SUMMARY

You learnt how to package your application as a release candidate, which consists of a collection of NuGet packages, in this chapter. **Our rules of thumb are:**

- Build once, deploy many.
- Use the build number as the official version number throughout.
- For package format, use NuGet.
- Using Azure Artifacts, archive your release candidates.
- Test suites that must run in a deployed environment are packaged.

Package and publish shared libraries through Azure Artifacts.

REVIEW QUESTIONS

THEORY QESTIONS

1. Explain the rules of thumb for release candidate packages.
2. How to create and use release candidate packages? Explain

OR

Explain the sequence of events in DevOps Pipeline.

3. How to define the bounds of a package? Explain
4. Explain Azure Artifacts Workflow for Release Candidates.
5. Explain how packages are created.
6. How a build or release configuration can obtain a NuGet package from Azure Artifacts? Explain.

OR

How use release candidate package in deployment?

MULTIPLE CHOICE QESTIONS

1. _____ is typically made up of collection of packages.

a) Release Client b) Super User

c) Release Customer d) Release Candidate

2. A _____ file is a .NET package format

a) .* nupkg b) .nupkg c) * nupkg d) nupkg

3. _____ generates release candidate package, which are then used by the deployment configuration

a) Private Build b) Public Build

c) CI Build d) Protected Build

4. Which of the following is not an application deployable component?

a) Oracle Server Database b) ASP.NET Web Application

c) Off-line Job d) SQL Server Database

5. _____ test is only found in the TDD environment

a) Integration b) Unit c) Acceptanced) Full-System

REFERENCES

- Duvall, P. M. (2007). Continuous Integration: Improving Software Quality and Reducing Risk. Addison Wesley.
- Jeffrey Palermo. .NET DevOps for Azure a Developer's Guide to DevOps Architecture the Right Way, Apress (2019)
- Retrieved from New Signature: https://newsignature.com/articles/all-about-azure-artifacts/
- Retrieved from Azure Info Hub: https://azureinfohub.azurewebsites.net/Service?serviceTitle=Azure%20Artifacts
- Retrieved from Tutorials Point: https://www.tutorialspoint.com/software_testing_dictionary/release_candidate.htm

12

DEPLOYING, OPERATING AND MONITORING THE RELEASE

Unit Structure :

12.0 OBJECTIVES

In this chapter you will learn the following concepts:

- Deployment pipeline's design model
- The various types of environments that can be configured within.
- The various types of activities necessary during deployment.

12.1 INTRODUCTION

This chapter mainly introduces the model for designing your deployment pipeline, the types of environments to configure within it, and the types of activities required during deployment. You will also learn about the many types of data that can be deployed or provisioned with a

deployment, as well as the various Azure alternatives for running code in Azure PaaS services or elsewhere. Finally, you will observe the many touch points in the release configuration, such as the effect of variables on the deployment steps' execution. It also focuses the entire DevOps process on .NET for Azure.

12.2 DESIGNING YOUR DEPLOYMENT PIPELINE

- You must decide how many environments to configure and the difference between them in order to define the proper structure of your deployment pipeline.
- There are following some principles that will guide those decisions:

1. **Build one, deploy many:**

- Regardless of how many environments you have, you'll deploy the same release candidate, which is generated from a single continuous integration build, several times, at least once per environment type.
- Once a version has entered deployment processes, do not rebuild or recompile from source.
- Consider the release candidate dead if there is an issue somewhere, fix the problem, and switch to a different versioned release candidate.

2. **Do nothing on production for the first time:**

- Design the deployment pipeline so that each unique activity in your deployment is tested in at least one pre-production environment before being run in production.
- For example, if your production environment is based on a web farm with multiple batch job servers and a huge SQL Server cluster, single-server configurations in all pre-production settings are not recommended.
- After the release candidate has been packaged by the continuous integration build, no files destined for production should be generated or altered.
- At the end of the CI build, put everything needed for production deployment into the release candidate packages.
- If you notice that something is missing, stop the line, add the missing element to the code base, and let the CI build package up a new release candidate that includes everything.

3. **Shift left on pipeline capabilities:**

- Push logic into script files stored in the application's Git repository when deploying application components and configuring settings and data.
- While the CI build configuration and deployment stages allow for the execution of scripts saved in any location, this strategy introduces global and temporal dependencies.
- As many commands, scripts, and logic as possible should be sourced from the Git repository and the packages that make up the release candidate that is being deployed.

12.2.1 Determining Environments

- Server environments are referred to by many different terms in our industry. Everyone has the ability to produce. The following are some other applications:
  - Local
  - Sandbox
  - Dev
  - Integration
  - Test
  - User
  - UAT
  - QA
  - QC
  - Acceptance
  - Staging
- While no standard exists, development, testing, staging, and production appear to be the most well-known pre-cloud services.
- In a DevOps environment, the team will always have at least three (3) deployed environments.
- The following table shows the 3 different types of environments in a DevOps Pipeline

Distinct Environment Types			
Environment Attribute	TDD	UAT	PROD
Purpose	Automated Verification	Manual Verification	Ongoing Operations
Audience	Engineering Team	Internal Customers	External Customers
Quality Control	Acceptance Testing	UX Testing	Automated Alarms
	Env. Recreation	Exploratory Testing	Health Checks
	Health Checks	Database Migration	Tracer Bullets
	Database Recreation	Health Checks	
	Security Testing	Observability Validation	
	Scale/Perf Testing	Other Manual Testing	
	Reliability Testing		
	Other Test Suites		

- The above table shows the 3 types of environments that you will require when designing your deployment pipeline.
- Production is something that everyone is familiar with. It exists for the benefit of the people who benefit from your software.

- Any form of manual testing should be done in the next environment before production. We'll call it user acceptance testing (UAT) because it's more about the users than the engineering team.
- Finally, we have a dedicated environment for all types of automated verification. We call it the TDD environment, short for test-driven development, to avoid confusion with other environment names that have been used in the past.
- Humans are not permitted in the TDD environment. If a human attempted to use this environment, they would discover that it was being created and destroyed at a rate that made any useful use impossible.
- Let's look at some examples of how to decide how many of each type to choose.

▪ **Production**

You can either provide a dedicated production environment for each customer or have all of your customers use a single production environment in the case of production.

▪ **UAT**

If your company is tiny, you can have just single manual verification environment. Alternatively, you may have several separate user or stakeholder groups who would benefit from having their own dedicated environment so that they may select when to accept the next release candidate that is available.

▪ TDD

- This kind of environment is suitable for fully automated construction and destruction.
- Every successful build should result in a new deployment to this environment.
- Because you may have numerous feature branches active at the same time, your CI build should be parallelizable - that is, multiple builds should be running simultaneously, one for each active branch.
- Because each build triggers a deployment to this environment type, you can construct numerous instances of it at once.
- If you and a colleague both commit changes to your feature branch at the same time, for example, you want the build, packaging, and deployment to your TDD environment to happen promptly without having to wait on your colleague.
- This is accomplished by having the environment's name parameterized by your build or branch, as well as establishing a TDD environment object devoted to your build. The environment is then deleted after your acceptance tests have completed (pass or fail).

12.2.2 Assigning Validation Steps to Environments

- You decide how many actual environments to include in your DevOps pipeline.

- You'll never have less than three, but you might have more depending on how many of each type you choose.
- It is also up to you to decide which environments should be placed in series and which should be placed in parallel.
- If two stakeholder groups, for example, require their separate unique UAT (manual verification) environment, you can determine that they can each receive the new release candidate at the same time and validate it in parallel.
- In this case, you'd create two environments (or keep one around for a permanently) and deploy to each of them at the same time.
- Before deploying to production, you would wait until each group had validated the release candidate.
- The following table shows each environment type is built for different deploy and validation steps

Environment Deploy Steps			
	TDD	UAT	PROD
Migrate Database	✓	✓	✓
Deploy Application	✓	✓	✓
Run Health Check	✓	✓	✓
Load Static Data	✓	✓	✓
Load Test Data	✓	On-Demand	✗
Recreate Environment	✓	On-Demand	✗
Recreate Database	✓	✗	✗
Run Acceptance Tests	✓	✗	✗
Deploy Unattended	✓	✗	✗
Destroys Environment	✓	✗	✗

- We execute fewer steps as we go from automated validation (TDD) to manual validation (UAT) to production.
- The flow through the environments is designed to front-load as many validation checks as necessary in order to detect errors as quickly as possible.
- “Shift left” is a value statement that has grown in popularity in the DevOps community.
- The purpose of Shift Left is to create a process that finds as many errors as possible early in the process.
- When you need to recreate the environment or reload test data, the UAT environment includes certain on-demand options, but we'll always need to deploy the new version of the application and migrate the database.

12.2.3 Deploying Data Changes across Environments

- In DevOps, the database provides some unique challenges.
- There is no state in application components. They are easily destructible and reassembled.
- Data must be stored for years in storage components.
- The same concepts and principles apply to any data storage, whether it's a relational database engine, blobs, tables, json collections, or simply a directory of files on a network share, when we discuss about "the database."
- This data must be protected and maintained across a large number of application component deployments.
- While maintaining the database's integrity, the schema, or the structure within which the data is organized, must be upgraded and updated on a regular basis.
- The following figure 12-1shows different types of data to be managed in DevOps environment:

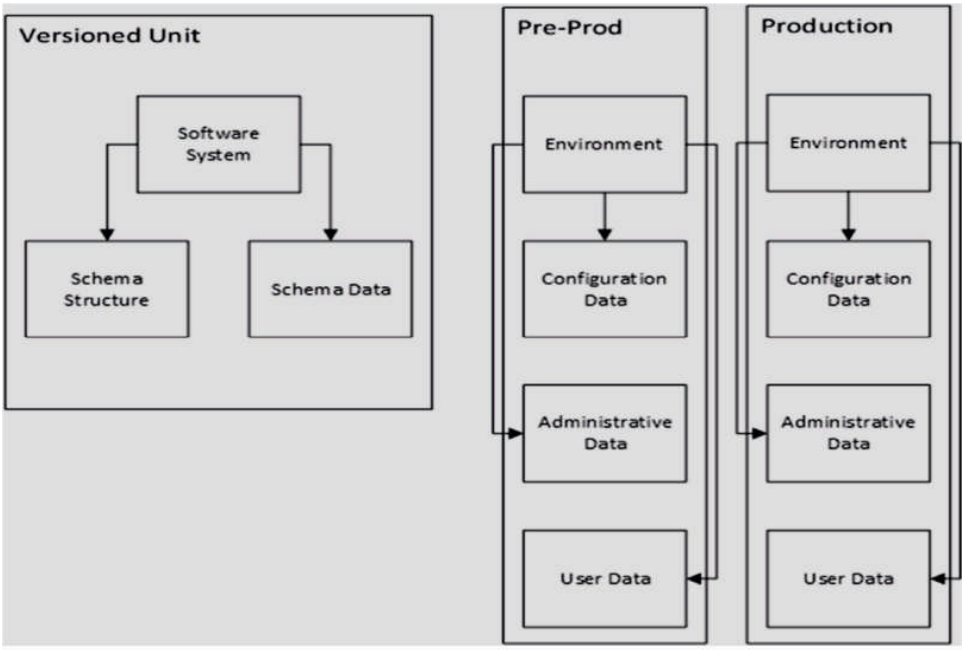


Figure 12-1: Each of the four unique types of data is managed in different ways

- All data and data concerns in a software system can be divided into four categories. In our DevOps pipeline, we manage these categories differently. They are as follows:

1. **Schema**

- The software system owns the schema, or data structure, and it should be consistent across all environments.
- This contains stored procedures, views, indexes, functions, and other SQL Server objects. Versioning and storing the schema with the application code is recommended.

2. **Schema data**

- This data is part of your schema's architecture and should be consistent across environments.
- Standard lists, for example, fall within this category.
- These lists can be used to create drop-down boxes in your software. Mr., Mrs., and other common name prefixes are a good example. During the development process, these are defined.
- This schema data should be created and deployed concurrently with schema updates, and it should be saved in the same version control system as the application code.

3. **Configuration data**

- The environment owns the configuration data. It should not be saved with the application code because it may vary depending on the environment.
- Some of it, such as passwords, tokens, and credentials, may be sensitive. An XML or JSON configuration file may contain some of this configuration data. Other configuration information could be saved in a database table.
- The data's nature does not change as a result of its storage location, nor does it change from one environment to the next. As a result, it should be deployed in the environment at the same time as the application and database.
- Whether it's putting a string into an XML file or inserting a record into a SQL Server database table, the automated deployment process should handle the process of getting the configuration data for the environment and properly deploying it.

4. **Administrative data**

- The organization that supports the environment owns the administrative data.
- Top-level user accounts or customer header records are two examples of this.
- In many applications, if there isn't even a single user account record, the software won't be able to do anything. To enable functions to light up, at the very least, a global administrator record may be required.
- Administrative data can, but does not have to, differ by environment.
- It may be the same in two environments but different in another since it is determined by the organization that supports the environment.
- Because this data is likely to contain credentials, it should be deployed to the environment automatically and not maintained with the application code.

5. **User data**

- The users who create the data are the owners of the data. It differs from one environment to another.
- This is the type of information with which you are most familiar. As more people utilise the system, it continues to expand and change.

- It should be maintained over from deployment to deployment. The integrity of user data is secured by all automatic database migration processes and tools.

12.2.4 Choosing Your Runtime Architecture

The logical architecture of application as shown in following figure 12-2:

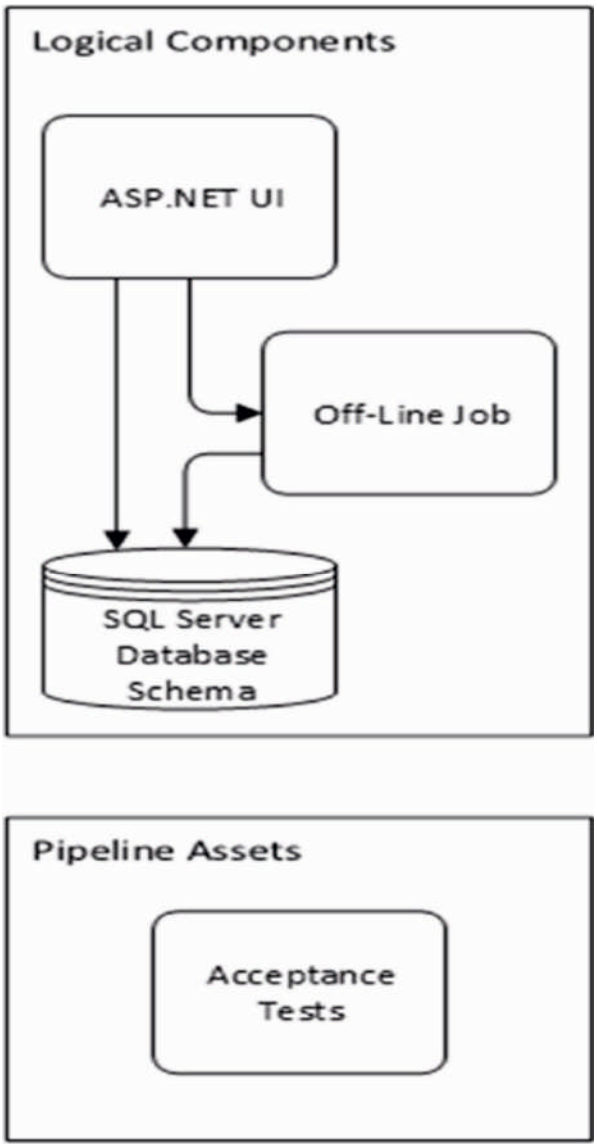


Figure 12-2: The Logical Architecture of Application

- At its most basic level, our application consists of three logical components, as well as an acceptance test suite that must be deployed in some way in order to run against the application in a TDD environment.
- When it comes to designing an appropriate environment with this application, we have a lot of alternatives, as shown in the following figure 12-3

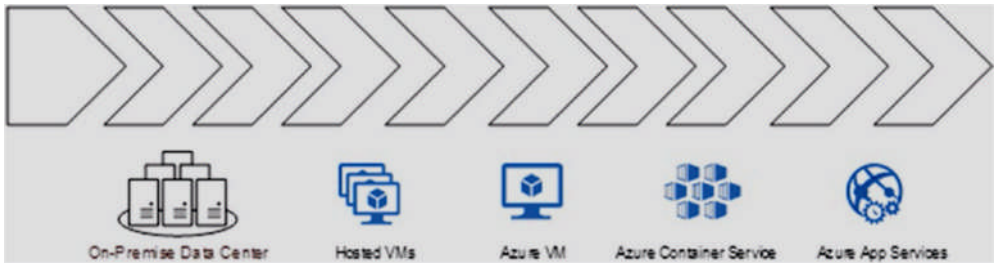


Figure 12-3: Each application can be deployed on a spectrum of environment types

Explanation:

- As seen in above figure there are various options are available to deploying our application. The left-hand side options provide us more control, but they also come with more responsibility and maintenance.
- The options towards the right side limit the scope of computing resources that we can manage, but they also relives us of more responsibility and maintenance.
- We are responsible for less maintenance because we control less of the computing environment.
- As the options move to the right, your application will have fewer APIs and resources available.
- For example, if your web application, utilizes a custom font to display a screen, it will be incompatible with Azure App Services, which do not support font installation on the underlying servers. However, if your application just uses APIs available in that environment, it's the most low-maintenance option to run your web app and off-line job.
- We've already determined that we won't be installing physical servers in a cabinet in our own data centre, but that option would work perfectly for our application.
- We could work out a deal with a local hosting company to have some virtual servers set up for us. We'd also set up several virtual machines on Azure.
- We'd probably set up a few web servers, one or more servers for the off-line task, and a SQL Server cluster for the database. While our acceptance test suite is running, we can probably use any server to host it.
- We can use containers or PaaS (Platform as a Service) in Azure if we don't want to operate a server operating system. Windows containers are developing, but they still have certain issues. If you're targeting Linux with your.NET Core applications, Linux containers are a better solution.
- PaaS services, such as Azure App Services, go even further than containers. Web applications, off-line jobs, and a container image can all be hosted on these servers.

The following figure 12-4 shows the physical architecture has been specified for our application:

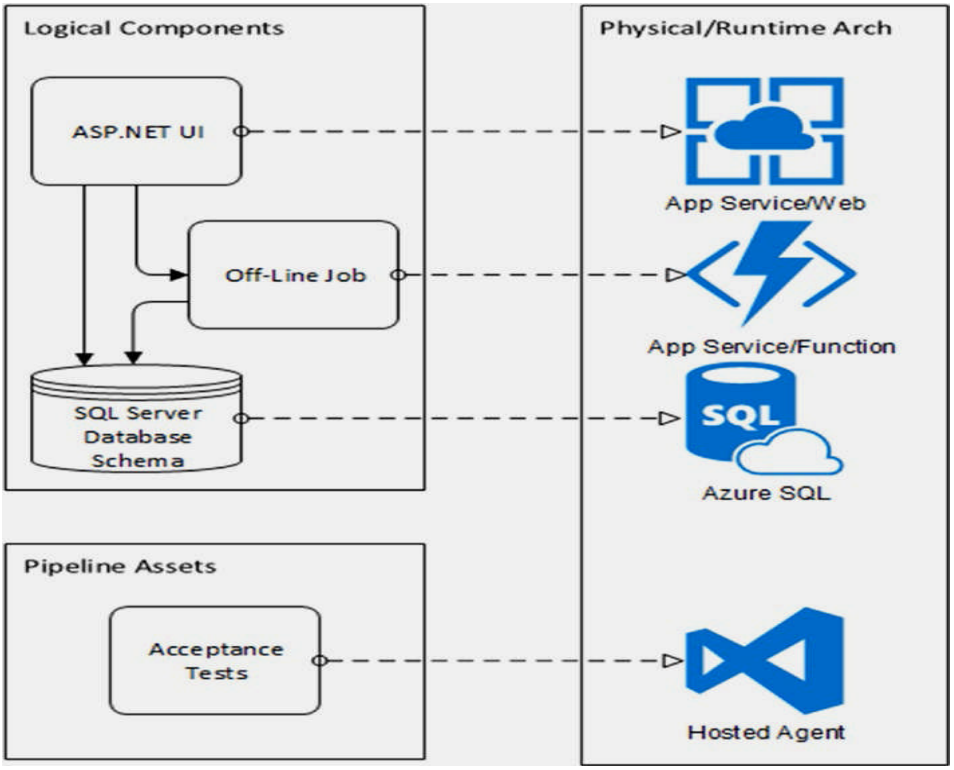


Figure 12-4: Physical architecture has been specified for our application

Explanation:

- For the **ASP.NET UI**, which is a web application, we've used **Azure App Services**.
- The **off-line job** will be hosted in App Services and delivered as an **Azure Function.**
- **Azure's SQL database** service will host the **SQL Server database**.
- The **acceptance tests** will be deployed to the **Azure Pipelines** hosted agent.
  From there, the tests can execute.

12.3 IMPLEMENTING THE DEPLOYMENT IN AZURE PIPELINES

In this topic we extend our pipeline from CI build and configure deployments across 3 environments.

The following figure12-5 shows the overview after it has been correctly configured.

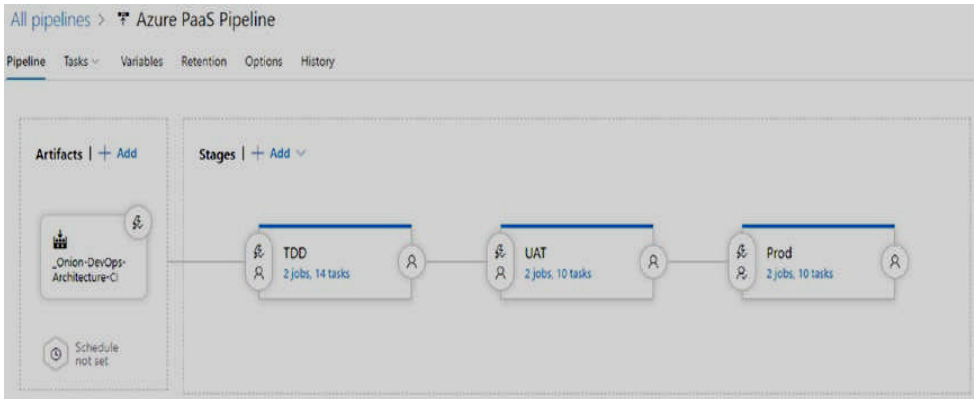


Figure 12-5: Our release configuration contains three environments and is triggered from the CI build

There are 4 key parts to our pipeline's release configuration. They are as follows:

1. **Artifacts:**
The release must be aware of the artifacts that are available to it. The tool has various options, but this is where you'll specify the build configuration that represents your application's CI build. You'll set up the release to start automatically whenever that build is completed.

2. **TDD stage:**
Multiple stages can be present in the release, which might be sequential, parallel, or both. For any of your apps, this is the smallest, shortest pipeline you'll have. The TDD stage corresponds to the fully automated TDD environment where your automated full-system acceptance tests run.

3. **UAT stage:**
The deployment of the application to the UAT environment is represented by the UAT stage.

4. **Prod stage:**
The deployment of the application to the Production environment is represented by the Prod stage.

After that, we'll move over each of the screens that require some sort of configuration.

1. **Artifact**

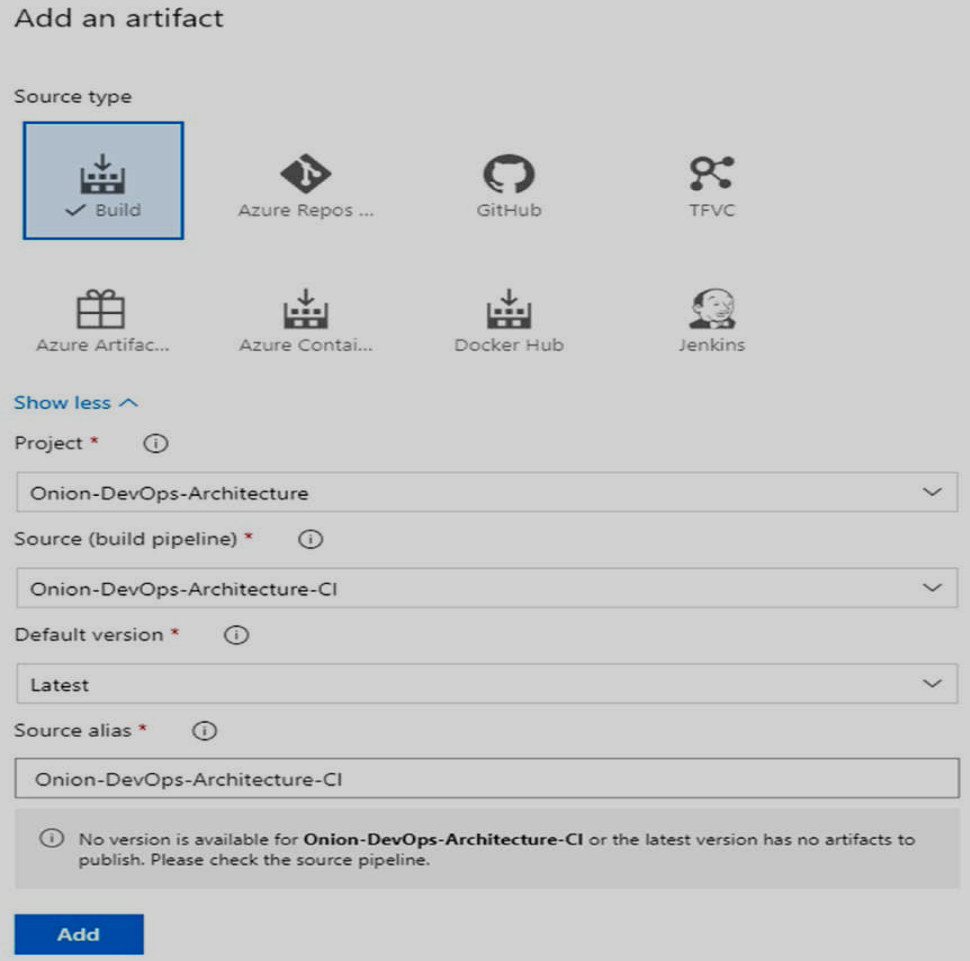


Figure 12-6: Specify the CI build that will be triggering the release

- The settings needed to wire up a CI build with an auto-triggered release are shown in above Figure 12-6.
- Use the default version of "Latest" to ensure that your release configuration works with any build from any active branch.
- You can keep a single CI build setup and a single release configuration this way.

- **Two important settings** can be found on the **build artifact's property page**:

1. Enable creating a release every time a new build is available.
2. Build branch filters setting: In that the drop-down menu appear, Put the mouse pointer in it and type an asterisk (*). This ensures that builds from all branches will cause the release to be triggered.

- This is required in order to perform acceptance tests and deploy the pipeline to the TDD environment.
- As shown in the following Figure 12-7, each stage/environment also includes branch filters to prevent branch-based release candidates from proceeding further down the pipeline.

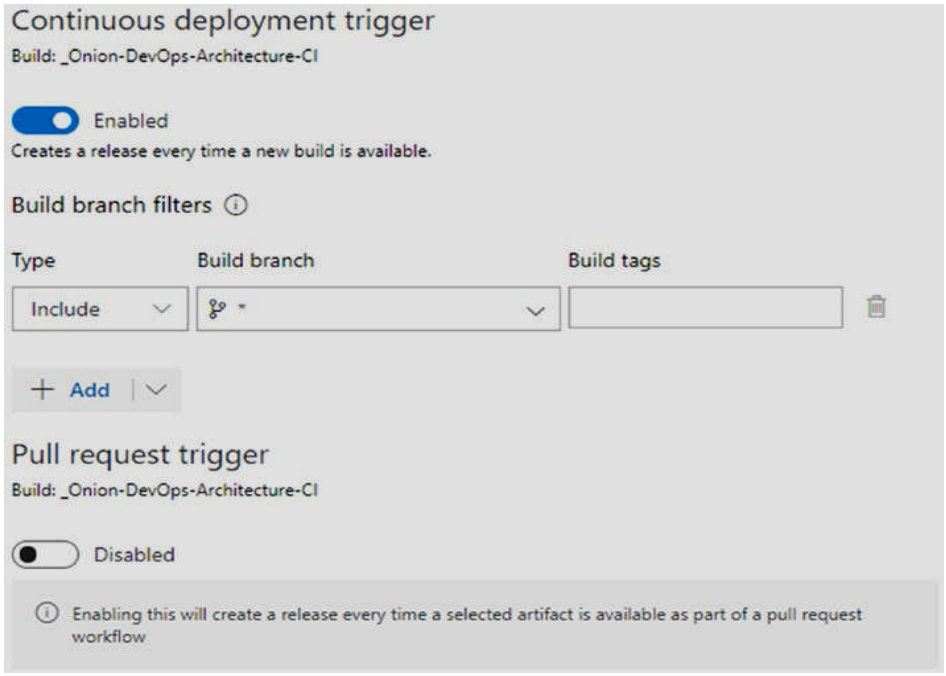


Figure 12-7: Enable continuous deployment to automatically trigger the release

2. TDD

- Leave all of the settings for the TDD environment at their defaults, and ensure that it triggers automatically after release.
- You only need to change this setting. Leave the default alone unless you have a reason to change them. Azure Pipelines contains filters and distinct logic points in several places.
- Because our application contains three bundled components, you'll need to configure the TDD environment in three sections. The website, the offline job, and the SQL database are all in place.
- This section gets the NuGet package for the application component you're installing. It correctly extracts and install that component before moving on to the next.
- Finally, you run the health check, which calls the necessary URL or API so that the application can run the built-in procedure that verifies that everything is started and online.

Multiple "jobs" can be configured for each stage (think environment). As shown in the following figure 12-8, our application has a full-system acceptance test suite that uses Selenium to control a web browser.

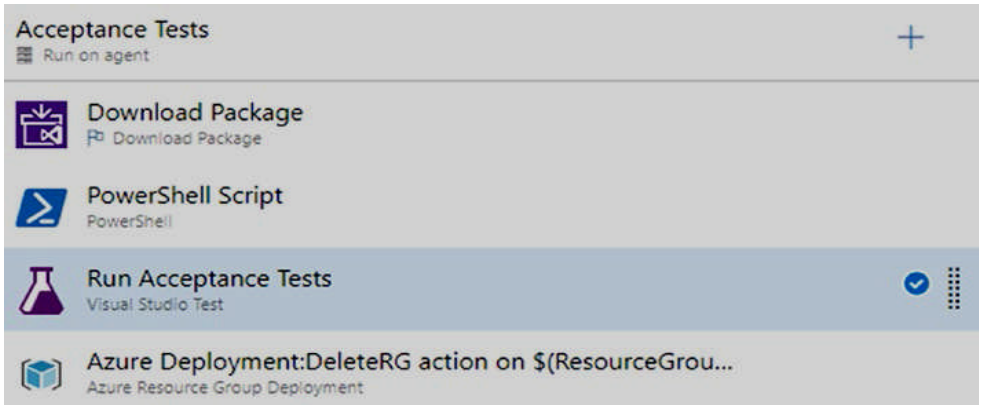


Figure 12-8: The second major task for the TDD environment is to run the acceptance tests

12.3.1 Deploying an Application Component

In this topic we'll focus on the most difficult component of most applications in this part. This is the SQL database. To deploy the database, we'll need all of the necessary assets on available, as well as the ability to establish the database in the TDD environment.

The four steps as shown in the following Figure 12-9 are responsible for on-demand SQL database provisioning and schema creation.

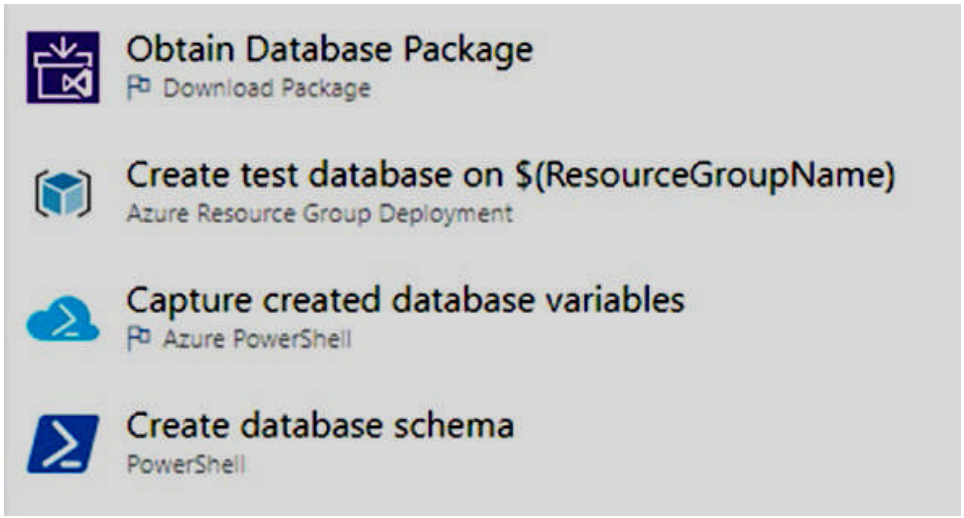


Figure 12-9: The four steps that make up a provisioning and deployment of the SQL database

1. **Download Package:**

- The custom properties of the Download Package task.
- This job downloads and extends the contents of the specified NuGet package from Azure Artifacts to the provided destination directory.
- We mention the complete name of the package: Onion Dev Ops Architecture. Database in order to get the NuGet package for the release candidate we want.

- After that, we must specify the version we want to obtain. We have the current $(Build. Build Number) available in this context, so we specify that.

2. Create the database on (Resource Group Name):

- The next step in our database deployment is to create the Azure SQL database in our TDD environment.
- We will specify the path to the ARM (Azure Resource Manager) template file.
- The Onion Dev Ops Architecture. Database NuGet package extracted this file for you, so you may use it.
- This ARM template is owned by the “Database” Visual Studio project and is stored in the Git repository.
- Some variables have been externalised as parameters in this ARM template so that the deployment process can control the settings. The text area "Override template parameters" can be seen.
- To make the ARM template generic and reusable across other applications that use an Azure SQL database, we specify several of these variables. We may control the database edition, size, and other parameters using these variables.

3. Capture created database variables:

- This step is used to create database. We have just created a new Azure SQL database, but we have no idea how to access it.
- Any databases must be housed in a new SQL Server database, which must have a distinct hostname.
- We loop through the outputs of the resource group deployment from the execution of our ARM deployment and save them as variables that may be used in further steps of our deployment.
- In this scenario, the result will be labelled "resource Group Unique String." As a result, we now have the server name, which we can use to run our schema migration tool.
- The following value is used to create the variable $ (Database Server):

Data base server $ (resource Group Unique String). database. windows. net

- This pattern is used by Azure SQL for database server hostnames. We may now access our newly created database server after capturing this variable.

4. Create database schema:

This shown in the following figure 12-10

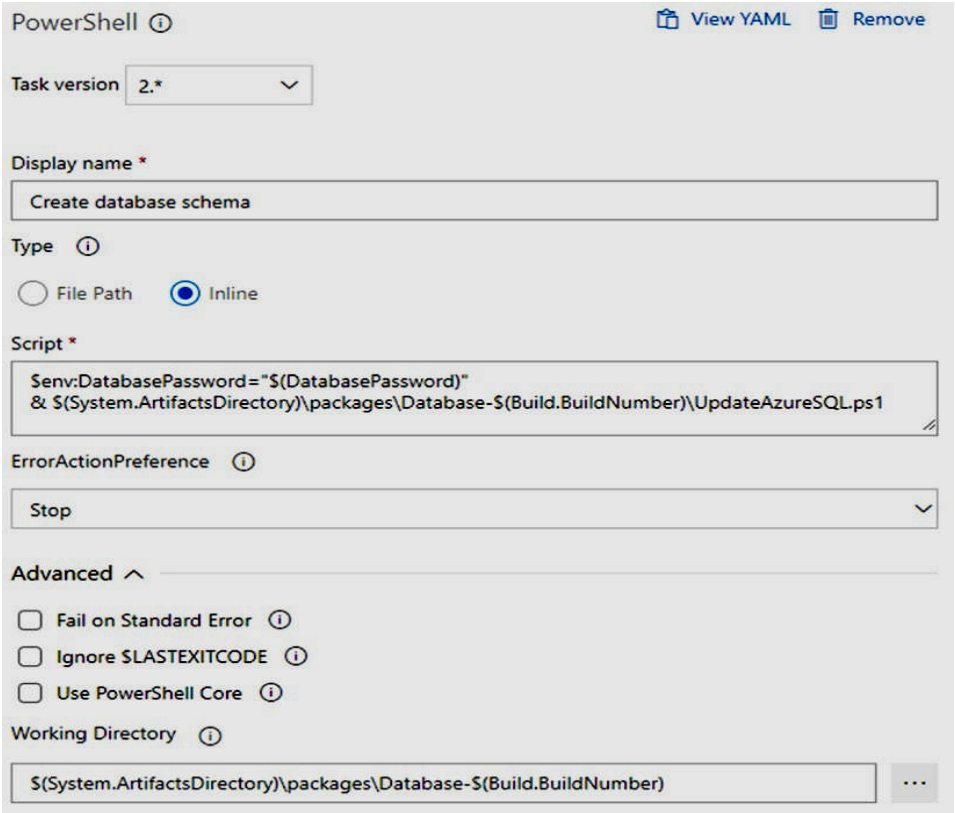


Figure 12-10. The step of our database deployment that creates the full database schema in the TDD environment

- We take the logic that needs to run and push it into our Visual Studio solution in this Power Shell task, which is another example of "shift left."
- We need to make the sensitive credential saved in the $(Database Password) variable available in order to run it.
- Environment variables are not created automatically when variables are marked as "secret."
- Our Power Shell snippet makes it available to the current process as an environment variable. Other variables in plain text are provided as environment variables by default.
- The working directory is another key setting for running Power Shell scripts that are included in your release candidate's NuGet package.
- As a result, our script will work in all of the places and environments where it may be executed.
- We specified the working directory to be the directory where our NuGet package was extracted, as shown in above Figure. The authoring and maintenance of the Power Shell script are easier as a result of this.
- Our TDD environment now has a complete SQL Server database with the whole schema and schema data loaded due to the execution of this

command. We now have the full connection string, which is ready to use.

- The other components of the application follow the same pattern:
- Retrieve NuGet package
- Extract NuGet package in a working directory
- Poke any configuration variables
- Provision server/cloud environment
- Install application component
- Start application component

12.3.2 Running Test Suites Using a Release Configuration

For running test suites, we use N Unit test framework with Selenium's web driver through the Google chrome browser. The listing of out test code given as below:

```
using System;
using System.IO;
using System.Reflection;
using ClearMeasure.OnionDevOpsArchitecture.Core.Model;
using ClearMeasure.OnionDevOpsArchitecture.IntegrationTests;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using Shouldly;
namespace ClearMeasure.OnionDevOpsArchitecture.AcceptanceTests
{
public class GetAllExpenseReportsTester
{
private string _appUrl;
private IWebDriver _driver;
[OneTimeSetUp]
public void Setup()
{
_appUrl = new DataConfigurationStub().GetValue("AppUrl",
Assembly.GetExecutingAssembly());
_driver = new ChromeDriver(".");
new ZDataLoader().LoadLocalData();
}
[OneTimeTearDown]
public void Teardown()
{
_driver.Close();
_driver.Quit();
_driver.Dispose();
}
[TestCase("000001",
```

```
TestName = "Should add new expense report numbered '000001'")]
[TestCase("000010",
TestName = "Should add new expense report numbered '000010'")]
[TestCase("000100",
TestName = "Should add new expense report numbered '000100'")]
[TestCase("001000",
TestName = "Should add new expense report numbered '001000'")]
[TestCase("010000",
TestName = "Should add new expense report numbered '010000'")]
[TestCase("100000",
TestName = "Should add new expense report numbered '100000'")]
public void ShouldBeAbleToAddNewExpenseReport(string
expenseReportNumber)
{
void ClickLink(string linkText)
{
_driver.FindElement(By.LinkText(linkText)).Click();
}
void TypeText(string elementName, string text)
{
var numberTextBox = _driver.FindElement(By.
Name(elementName));
numberTextBox.SendKeys(text);
}
Console.WriteLine($"Navigating to {_appUrl}");
_driver.Navigate().GoToUrl(_appUrl + "/");
_driver.Manage().Window.Maximize();
TakeScreenshot($"{expenseReportNumber}-Step1Arrange");
ClickLink("Add New");
TypeText(nameof(ExpenseReport.Number), expenseReportNumber);
TypeText(nameof(ExpenseReport.Title), "some title");
TypeText(nameof(ExpenseReport.Description), "some desc");
TakeScreenshot($"{expenseReportNumber}-Step2Act");
_driver.FindElement(By.TagName("form")).Submit();
TakeScreenshot($"{expenseReportNumber}-Step3Assert");
var numberCells = _driver.FindElements(
By.CssSelector(
$"td[data-expensereport-property=\"{nameof(ExpenseReport.
Number)}\"]
[data-value=\"{expenseReportNumber}\"]"));
numberCells.Count.ShouldBeGreaterThan(0);
numberCells[0].Text.ShouldBe(expenseReportNumber);
}
private void TakeScreenshot(string fileName)
```

```
{
var chromeDriver = ((ChromeDriver) _driver);
chromeDriver.GetScreenshot().SaveAsFile($"{fileName}.png");
TestContext.AddTestAttachment($"{fileName}.png");
}
}
}
```

- In the TDD environment, we use a Data Configuration Stub() to clear out the database and preload it with a few records.
- Six test cases are run using the same N Unit test. The steps in his test progress are as follows:

1. Go to the home page.
2. Locate and click the "Add New" button.
3. Type a value into the Number text box.
4. Type a value into the Title text box.
5. Type a value into the Description text box.
6. Fill out the form and submit it.
7. Locate the table's row and the Number column.
8. Verify that the Number's value is the expected value.

- This test will open the local Chrome browser on the server and run these steps as it runs. Let's have a look at how Azure Pipelines configured this.

- After you've downloaded the NuGet package containing our acceptance test suite, follow these steps:

1. On our Azure DevOps agent server, we extract the package to a working path.
2. Configuration settings for ConnectionString and AppUrl are added to the test suite's configuration file.
3. Our *AcceptanceTests.dll assembly, which contains our tests, is used by the VSTest task.
4. There is no fourth step because we use VSTest, which means the test output is automatically captured as a test run by Azure DevOps.

- Let's look at the NuGet package for our acceptance tests first, as illustrated in the following Figure 12-11.

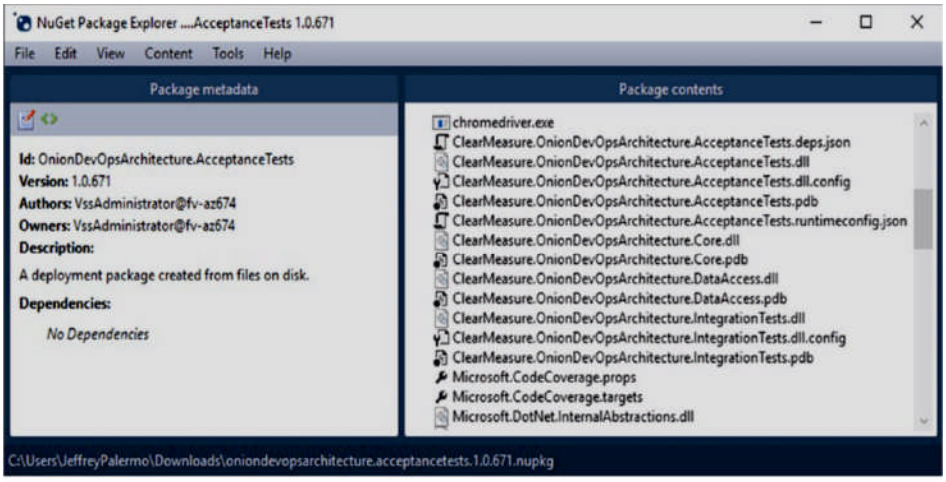


Figure 12-11. The acceptance test package contains the Selenium driver as well asthe test assemblies

- In above figure12-11 notice that the Chromedriver.exe, as well as the test assemblies and config files that go with it, are all included in the package.
- All test runs, tests, and their results are kept in Azure Test Plans. Furthermore, each test that is run has the ability to archive any arbitrary file attachment.
- One of the most valuable attachments for full-system acceptance tests that run through a browser UI is a screenshot of every screen the test views as it runs.
- Our C# test scenario includes calls to the Chrome Driver to capture a screenshot, which we then save and connect to the Test Context.
- When VSTest performs these tests, it gathers all of the data and saves it to Azure Test Plans.
- Let's take a step-by-step process to this. First, we can view the results of our continuous integration build.
- On this build summary page, shown in the following Figure 12-12, we can see that the build was successful, as well as the deployments to TDD and UAT. We can also see that the Prod deployment is ready to go.

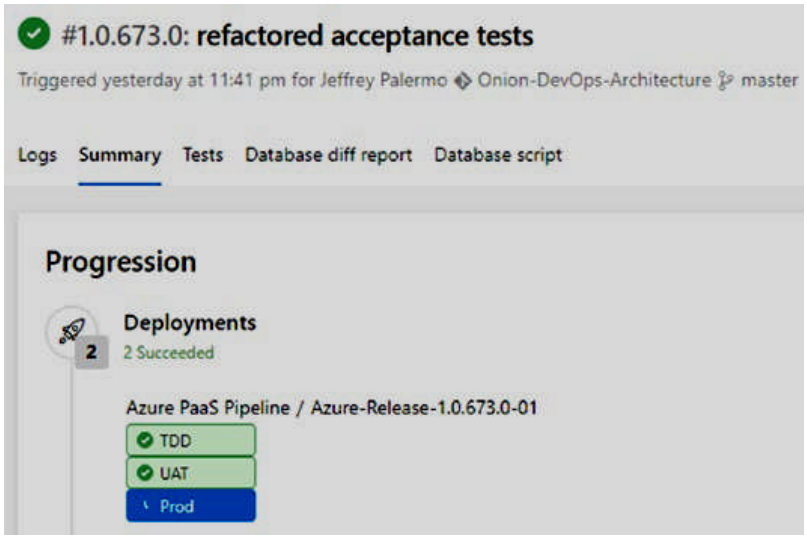


Figure 12-12. *The build summary page shows that this build has been deployed across environments*

- We can see information about the release and go into each environment to see details about what happened if we click to our TDD deployment from the build page, as shown in the following Figure 12-13.

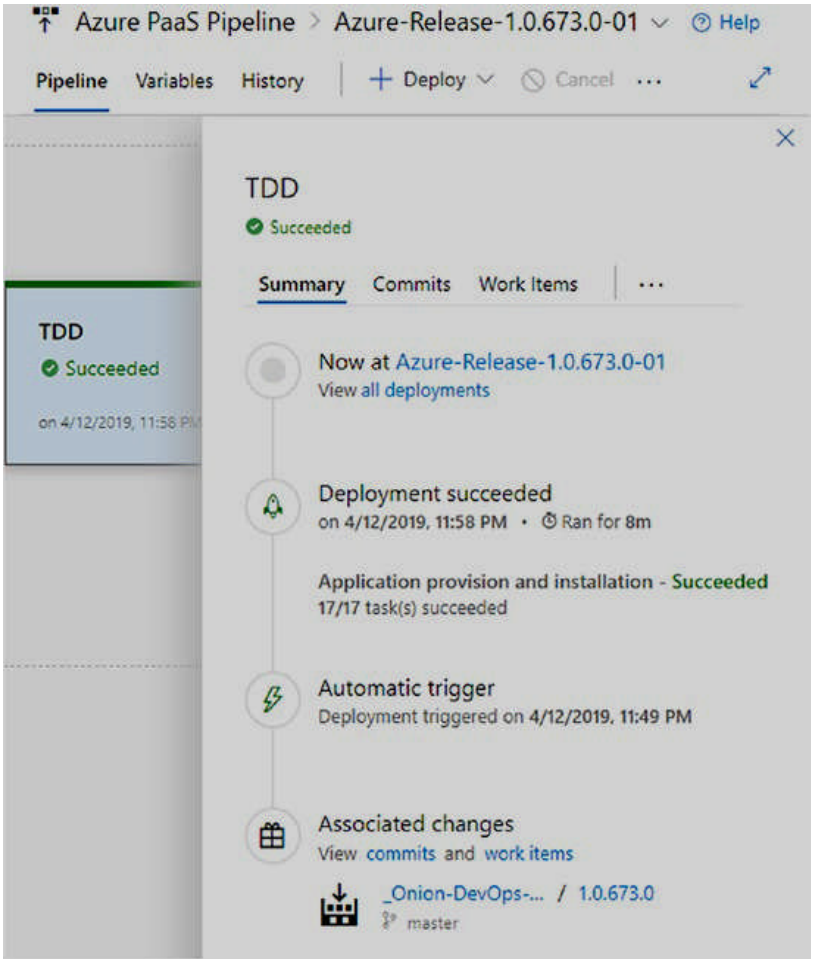


Figure 12-13. *The TDD release view shows the top-level details of the TDD Deployment*

- The Tests tab gives us access to the acceptance tests that have run, in addition to looking at the Logs of your deployment, which is crucial in debugging it until it works properly.
- We have six tests in our case. Select and click the last test on the list, "Should add new expense report numbered '100000'."
- When we select a test, it gives us more information about that test and the run of it.
- The remaining deployments in your pipeline will follow suit once your TDD environment has been created, configured, deployed, and tested.
- On the TDD environment, a 100% will appear next to the beaker icon. This indicates that a test suite was run and that all tests were successful. If any tests were skipped, the result would be less than 100%.

12.3.3 Differences in the UAT and Production Environments

- While the deployment process for the TDD environment should be similar to that for UAT and Prod, there will be some major differences in order to keep all of the trunk-based development branching possibilities.
- User Acceptance Testing (UAT) is a term that refers to the process of determining whether or not the term user acceptance testing (UAT) refers to a user/client testing software to see if it can be accepted or not.
- This is usually the final stage before the product goes live or before the product is accepted for delivery. After the product has been completely tested, UAT is performed.
- Data is moved to PROD Server after UAT is completed. Because it is the environment that consumers directly interact with, the production environment is often known as live, especially for servers. This is where real-time data is kept.
- It is necessary to configure the UAT state deployment to ignore release candidates created by feature branches.
- Only the master branch's release candidates should be deployed in the UAT environment.
- Variable differences, not process differences, are the focus of the UAT stage.
- We do not rebuild the database in the UAT environment. Rather, we keep the database and data intact while updating it by running just the *.sql files that haven't been run in that environment before. The value "Update" represents this.
- When this is completed successfully, we can be confident that the data will be maintained and the schema will be changed correctly when we perform the same routine in production.
- Next, look at the various configuration options for the Production deployment step.
- An individual, a group, or multiple individuals can approve a release stage. There are various options available to you.

- You can also enable the Gates feature, which allows you to bring in some business logic to assess whether or not the deployment should be permitted to proceed.
- If a prior release is in the approval queue, the new release will be held up until it is approved.
- The status will auto-update on the screen after the production deployment has been approved.
- If the deployment succeeds, it means that all components of the application, as well as their dependencies, have been deployed and that everything is online and working properly.

12.4 OPERATING AND MONITORING THE RELEASE

- We've only just begun after our software improvements are deployed and running in production. Consider our DevOps model.
- We'll need an operational strategy. Then we must continuously execute that approach and measure to ensure that what we expect is occurring.
- Learning as a feedback into planning for future modifications is emphasised in our DevOps cycle around the outside.
- By operating the software with real customers using it, we can

1. Verify that our customers can achieve their objectives.
2. Figure out what the best next change is.

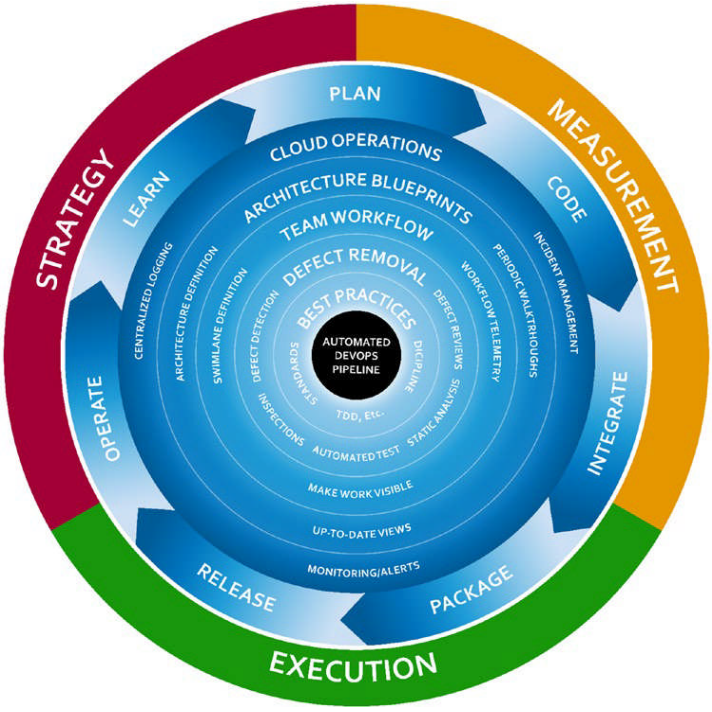


Figure 12-14. Onion DevOps Architecture provides a model for a complete DevOps Environment

- The release stage is just past the halfway point of the cycle around the outside of the onion.
- We must operate software well after it is released to our clients, learn how it performs, and then integrate that information back into future plans.

12.4.1 Principles

- A desired feature of a software system has been known as **testability** in discussions about quality and testing.
- **Observability** is a desired property for a software system operating on Azure, or any environment for that matter.
- **Observability** is tooling or a technical solution that allows teams to actively debug their system. Observability is based on exploring properties and patterns not defined in advance.

- **The principles are as follows:**

1. **Know what your software is doing at all times**

- It is insufficient to know that a server is operational or that a website is accessible.
- A client is down if any system function no longer performs as expected.
- We should consider our customers and their objectives.
- Customers are down if they are unable to complete tasks, even if the technical aspects of our system are operational.
- With that Perspective, we might question ourselves, "What do we need to know to be confident that our customers are up?"

2. **Listen to what your system is saying**

- You can have your software system emit and speak using many types of telemetry.
- Pay attention to what the system is requesting. When presenting to the Azure DevOps User Group, Eric Hexter, a visionary in DevOps Diagnostics, said that the system might ask for non-functional features or maintenance using telemetry.
- We can find out what work needs to be done on the system by looking at the logs and metrics. This work may not appear in standard product backlogs.
- Customers sending trouble tickets or problem reports is one evidence that the system's observability sophistication is insufficient.
- Consider the work in this field as a form of insurance policy. In order to be successful in this field, you must commit time, effort, and some money in products.
- For the return on risk avoidance, an insurance policy contains a premium that must be paid.
- If this is ignored, and the insurance premium is not paid, your company will be responsible for the entire cost of a business interruption.
- This is another example of the "Shift Left" way of thinking, where we may build observability into the system to improve customer service.

12.4.2 Architecture for Observability

- Observability helps developers understand multi-layered architectures: what's slow, what's broken, and what needs to be done to improve performance.
- There are 3 types of telemetry that should be emitted from software. They are as follows:

1. Metrics/performance counters

- Many of these are incorporated into the Azure platform, but you'll want to record queue length, for example, if you're using queues.
- Another valuable measure is the number of users by type who have used your system in the last hour or day. Trends like these can be utilised to set up alarms.
- If your normal usage suddenly reduces, for example, you may be experiencing a technical issue that has to be investigated.
- A metric is a value that expresses some data about a system.
- These metrics are usually represented as counts or measures, and are often aggregated or calculated over a period of time.
- A metric can tell you how much memory is being used by a process out of the total, or the number of requests per second being handled by a service.

2. Log messages/log files

- A system's every operation or transaction should be logged.
- Additionally, log files and log messages from different components should be combined and centralised into a single repository that can be queried as a whole to provide a complete picture of what the system is doing.
- Logs are distinct records of "what happened" to or with a specific system attribute at a specific time.
- They are typically easy to generate, difficult to extract meaning from, and expensive to store.

3. Heartbeats

- Heartbeats can be generated externally or embedded directly into application components.
- These are built-in health checks in the form of signals and fake transactions.
- If a significant integration is between the application and a payment processor, for example, it's important to verify that the payment processor's connection is operational.

The following figure 12-15 shows the architectural model for observability in Azure.

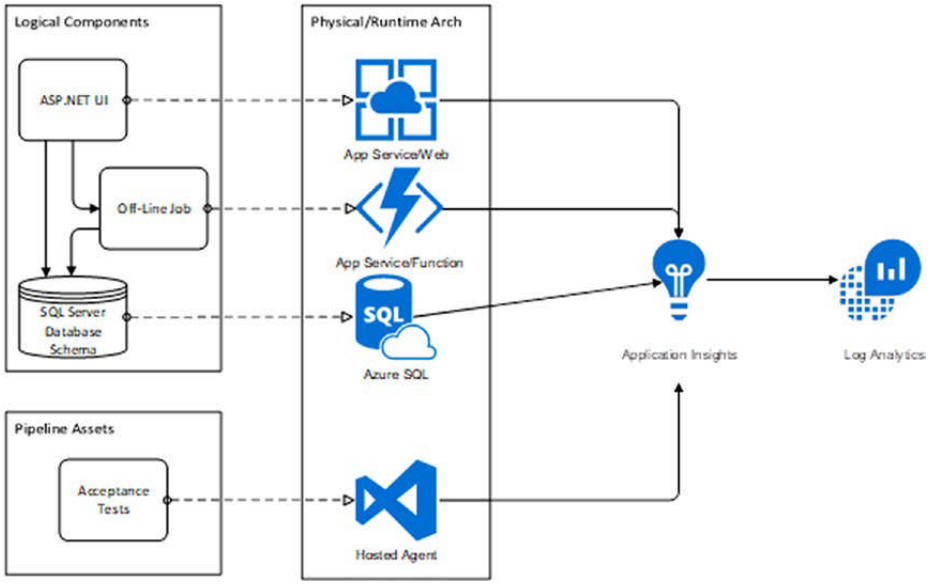


Figure 10-2. Each application should send telemetry and diagnostics information to a single Application Insights service

- Application Insights can collect a lot of data from every running component of your application, including the DevOps pipeline itself.
- Collecting all available information in one place is one of the ways we improve the observability of our software.
- This is something that Application Insights can help with. If you've never used Application Insights (also known as AppInsights).
- We have a stable production environment, one or more UAT environments, and a whole host of TDD environments that are constantly generated and destroyed, as you can see in our DevOps pipeline.
- The data in AppInsights should be able to resist changes in environments.
- **Consider the following environment architecture as shown in the following figure 12-16:**

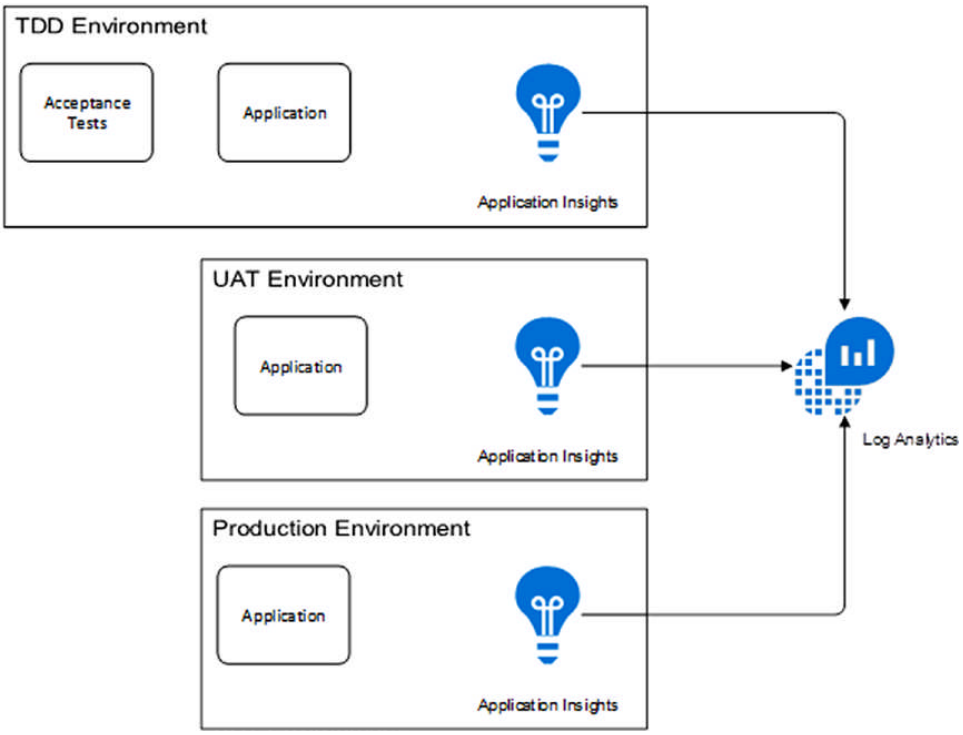


Figure 10-3. Each environment benefits from an AppInsights instance, which can then be aggregated to Log Analytics or other analytics sink

- We may customise queries for each environment type to the audience by giving an AppInsights instance for each environment type.
- Our TDD environment, for example, will have environments come and go as new builds are made.
- We might be able to detect whether transaction runtimes change by more than a particular percent by capturing performance metrics while full-system acceptance tests run.
- This could indicate a performance decrease from one version to the next. In addition, warnings should be configured on our production AppInsights repository.
- The objective of AppInsights is for each application to have its own AppInsights service.
- While custom tags can be added to telemetry to help filter out environments in a single AppInsights instance, the service was not built with that in mind.
- The service was created with the goal of collecting data from a single application. That is, one versioned, completely operational software unit.
- You can still use a single AppInsights service for the software system in production if your software is split into multiple Git repositories with multiple DevOps pipelines.
- We have multiple components in our sample application that will provide telemetry to a single AppInsights instance for each environment. AppInsights is divided into three services, one for each environment.

12.4.3 Jumpstarting Observability

- We want AppInsights to be durable while we're deleting and restoring TDD environments – and UAT environments from time to time.
- As a result, we separate the AppInsights services from the environments that may be deleted in a resource group.
- These services are stored in the “Onion-DevOps-Architecture-diagnostic” resource group. In contrast to your pre-production environments, this resource group will last a long period.
- After we've set up AppInsights, it's time to get the app ready to submit telemetry. First, install the AppInsights NuGet packages to your projects is visual studio.
- Then we will select the UI and Core. App Start-up projects to receive the AppInsights dependency.
- Then select the NuGet package i.e Microsoft. Application Insights. Asp Net Core. This package is suitable for Azure AppServices code, including WebJobs and Azure Functions.
- Once you've installed the AppInsights NuGet package, you'll need to locate an architecturally suitable location in your application to observe transactions so that the data can be sent to AppInsights.
- Expense Report Controller is an ASP.NET MVC controller action that contains the following code:

```
public IActionResultIndex()
{
var command = new ListExpenseReportsCommand();
ExpenseReport[] reports = _bus.Send(command);
var orderedReports = reports.OrderBy(report =>report.Number);
return View(orderedReports.ToArray());
}
//..//
public class ListExpenseReportsCommand :IRequest
{
}
```

- Also, make sure the appsettings. json file does not transfer telemetry from a workstation to an environment's AppInsight instance.
- The following is an example of the appsettings.json file:

```
{
"Logging": {
"LogLevel": {
"Default": "Debug"
}
 },
 "AllowedHosts": “*”,
"ApplicationInsights": {
"InstrumentationKey": "bogus value" } }
```

- The Application Insights Search window, allows you to develop your diagnostics capability locally without having to connect to Azure.

- When you run the application in debug mode, you can look at not just the telemetry you provide, but also the massive quantity of data that is automatically collected for you.
- You added the Custom Event, but nothing else to capture the SQL statement that was executed from the application to the SQL Server database.
- To understand more about your application, use this view to search across any number of attributes.
- You can use the command line to run your full suite of automated full-system acceptance tests while recording the telemetry in a debug session to explore further into:

```
dotnet vstest
    .\ClearMeasure.OnionDevOpsArchitecture.AcceptanceTests.dll
```

- You can gather a lot of telemetry to search if you execute the above command from the acceptance tests assembly's folder.
- Application Insights does not capture any data, parameters, or arguments automatically. As a result, you'll have to add code to do it directly.
- When you do, be sure that any sensitive data fields are exported to a monitoring system that may have different data security restrictions than the production database.
- Then, after you're confident that your application has a relevant iteration of telemetry, you can link it to the various environment-specific AppInsights services in your Azure subscription.
- It's critical to know when new release candidates were promoted from one environment to another for the execution of our deployment. **Add a Release Annotation task** to your deployment steps to accomplish this.
- This should be the initial job, so that even if the deployment fails, Application Insights gets a notification that one was started.
- AppInsights records that a version started deploying in this way, even if the deployment fails in the middle.
- You'll also add the appsettings.json file to the JSON variable substitution text section in the "UI Deploy" phase so that your variables are examined for JSON substitution.
- Your release configuration is ready to deploy once you've configured this and included a correctly named variable per environment with the Instrumentation Key.
- You'll be able to observe release markers and telemetry from each environment when your next release runs.
- You may obtain full information about the release that deployed the application that is responsible for the recorded telemetry by clicking the release marker within the Azure portal.

- The release marker provides links to the original release as well as the Azure DevOps project for the telemetry-emitting application.
- You've now connected code written in Visual Studio and stored in Git with live usage data recorded in Azure environments.

SUMMARY

You learned how to create your deployment pipeline in this chapter, and you got hands-on experience executing a correctly built release configuration. It's crucial to figure out the right quantity and type of environments. You also assigned each environment validation stages, including a built-in application health check. You learned about the many forms of data that may be distributed or provisioned with a deployment, as well as the various possibilities for running code in Azure PaaS services or elsewhere. Finally, you observed the many touch points in the release configuration, such as the effect of variables on the deployment phases' execution. You also learnt how to incorporate a full-system acceptance test suite into your TDD environment, for both feature branch and master releases. You have also learned the entire DevOps process on .NET for Azure

REVIEW QUESTIONS

THEORY QESTIONS

1. Explain the principles that decide how many environments to configure and the difference between them to define the structure of deployment pipeline.
2. Explain the 3 types of environments in a DevOps pipeline.
3. Explain the 4 categories in a software system.

OR

Explain the different types of data to be managed in DevOps environment.

4. Explain the logical operation of application.
5. Explain the physical architecture of application.
6. Explain the 4 key parts to our pipelines release configuration.

OR

Explain the implementation of deployment in Azure pipelines.

7. Explain the 4 steps of deploying an application component.

OR

Explain the 4 steps that make up a provisioning and deployment of the SQL database.

8. How running test suites using a release configuration? Explain.
9. Write the difference in the UAT and production environment.
10. Explain Onion DevOps architecture.
11. What is observability? Write its principles.
12. Explain the architecture of observability.

OR

Explain 3 types of telemetry that should be emitted from software.

13. Explain the concept of jumpstarting observability.

MULTIPLE CHOICE QESTIONS

1. In a DevOps environment, the team will always have at least _____ deployed environments.

a) Two b) Three c) One d) Four

2. Full form of TDD is _____.

a) Test Driven Deployment

b) Test Drive Development

c) Test Driven Development

d) Test Drive Deployment

3. Full form of UAT is _____.

a) User Accept Testing

b) User Accepted Testing

c) User Accepting Testing

d) User Acceptance Testing

4. TDD Environment is suitable for _____ automated construction and destruction.

a) Fully b) Partially c) Single d) Multiple

5. _____ is a value statement that has grown in popularity in the DevOps community.

a) Shift Right b) Shift Up c) Shift Down d) Shift Left

6. The purpose of Shift Test is to _____ a process that finds as many errors as possible early in the process.

a) Delete b) Create c) Move d) Update

7. Which of the following is not a unique type of data is managed?

a) Schema b) Production Data

c) Schema Date d) Configuration Data

8. The _____ system owns the data or schema structure, and it should be consistent across all environments.

a) Server b) Software c) Cloud d) Network

9. At its most basic level, air application consists of _____ logical components.

a) Two b) Three c) Four d) Five

10. Azure's SQL Database service will host the _____ server database.

a) MySQL b) SQL c) MongoDB d) Oracle

11. Which of the following is not the key part to our pipeline's release configuration?

a) Artifacts b) Development Stage

c) TDD Stage d) UAT Stage

12. The deployment of the application to the production environment is represented by the _____.

a) UAT Stage b) PROD Stage

c) TDD Stage d) Artifacts

13. Which of the following step is not responsible for on-demand SQL database provisioning and schema creation?

a) Obtain database package

b) Create test database on $

c) Capture created database variable

d) Create table

14. _____ step is used to create database

a) Obtain database package

b) Create test database on $

c) Create database schema

d) Capture created database variable

15. For running test suits, we use _____ test framework with selenium web driver through the google chrome browser.

a) Acceptance b) Integration c) Unit d) Back-Box

16. In TDD environment, we use a _____ to clear out the structure and preload it with few records.

a) Data configuration Clear ()

b) Data configuration Delete ()

c) Data configuration Remove ()

d) Data configuration stub ()

17. After the product has been completely tested, ____ is performed.

a) Unit Testing b) Integration Testing

c) User Acceptance Testing d) Acceptance Testing

18. Data is moved to _____ server after UAT is completed.

a) PROD b) Database c) SQL d) Oracle

19. _____ stage is just past the halfway point of the cycle around the outside of the onion.

a) Production b) UAT c) TDD d) Release

20. The desired feature of a software system has been known as ____ in a discussion about quality and testing.

a) Usability b) Testability c) Observability d) Capability

21. _____ is a tooling or technical solution that allows team to actively debug their system.

a) Observability b) Testability c) Capability d) Measurability

22. There are _____ types of telemetry that should be emitted from the software.

a) Two b) Three c) Four d) Five

23. _____ is a value that expresses some data about the system

b) Metric b) Metrix c) Count d) Unit

REFERENCES

- Jeffrey Palermo. .NET DevOps for Azure a Developer's Guide to DevOps Architecture the Right Way, Apress (2019)
- Retrieved from: https://answers.sap.com/questions/12151930/production-and-uat-server.html
- Retrieved from: https://cloud.google.com/architecture/devops/devops-measurement-monitoring-and-observability
- Retrieved from: https://lightstep.com/observability/

❖❖❖❖

Unit V

13

INTRODUCTION TO APIS AND API STRATEGY AND ARCHITECTURE

Unit Structure :

13.0 OBJECTIVES

The objective of this chapter is to enable the student to understand the basics of APIs with a practical explanation. This chapter also deals with API economy and helps the students to understand how APIs are used in the public sector.

This chapter also enables the student to in understanding API Strategy and Architecture.

13.1INTRODUCTION TO APIS

The following chapter gives an introduction to APIs. APIs or Application Programming Interfaces are important tools for businesses across all industries as they allow the functionalities of one software to be used by another. They are a means by which two different programs are able to communicate with each other. Figure 13.1 shows the functioning of an API as an intermediate between various applications.

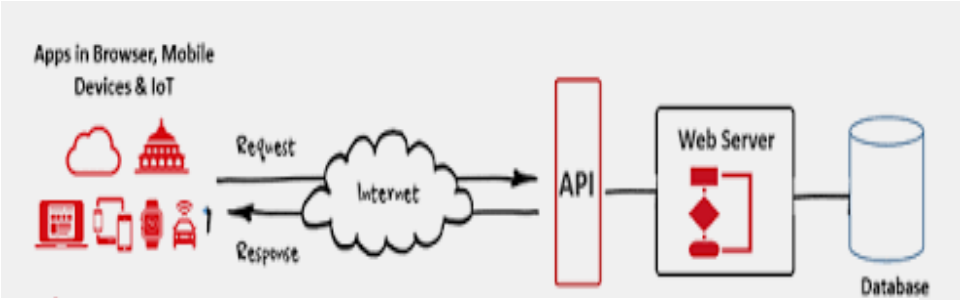


Fig 13.1 Source:https://apifriends.com/api-management/what-is-an-api/

A practical use of an API is when Ridesharing Applications like OLA or UBER use Google maps for completing a ride. What is essentially happening here is that an OLA or an UBER application is communicating with the Google Maps application. Here one application is using the services offered by another application. Another example of an API is when Travel sites gather information about flight details from Airlines web sites. Here the two applications are sharing data between them.

Application like GoIbibo and Trivago are able to offer competitive discounts on travel and stay because they are able to aggregate information from Airlines and Hotels etc. Without the use of an API this aggregation of information would not have been possible for these applications.

Application developers can embed video players into their site, reproduce reports, and access other helpful resources with the help of the You Tube. This prevents them from reinventing the wheel by using the functions of an existing application in their own application. This results in reduction of development time and costs.

APIs have also been instrumental is driving innovation in Technology and Science through collaboration and access to data. Research in Genomics got a shot in the arm when there was global collaboration and easy access to data through applications like the Google Genomics [1].

In short APIs are useful in

- Searching, collecting and sharing of data between applications.
- Eliminating redundancy by allowing businesses to make use of existing applications in their ongoing application development.
- Streamlining and integrating data into one's own systems, workflows, websites and products.

13.1.1 Types of APIs

APIs are of two types:

1. **Internal/private APIs**
   These APIs are accessible to developers and users within an organization. They are used to connect and easily integrate the different internal team processes.

2. **External/open APIs**
   Open APIs help external developers to easily access and integrate information from a tool developed by another developer or organization. External APIs save the developers time by allowing them to use previously existing tools within their applications, thus reducing the time and cost for creating these tools.

13.1.2 API Standards

Much like other technologies, APIs have also got to adhere to certain standards. These standards are used to establish how data is fetched and how APIs are accessed. The API standards are also called web service protocols.

Majority of the open APIs use one of these two protocols.
There are essentially two standards of APIs or web service protocols:

1. SOAPor the Simple Object Access Protocol
2. REST, which stands for Representational State Transfer.

SOAP was used extensively until REST came into the scenario. REST became popular because it offers a greater number of data formats. REST is also easier for developers to access and offers faster load times and better performance [1].

13.2 Practical implementation of APIs

APIs can be implemented in two different ways:

1. Based on language constructs or in the form of libraries/frameworks
2. As Systems exposing data and operations.[2]

13.2.1 APIs based on language constructs

While writing APIs programmers use language constructs like interfaces which are called by external parties.

```
internal Interface ILogin{
        Func<int>LogPageAsync(Page p);
        }
```

One of the implementations for the above interface can be:

Implementation**:**

```
        public class Loggingin : ILogin
{
 public async Func<int>>LogPageAsync (Page p)
```

```
{
Page.UserId =  SessionProvider.GetUserID();
// rest of the code}
```

Here the interface is using its own way of getting the value of User ID through a Session Provider as shown in bold in the above code. The above code works because the implementation gets the User ID from the Session Provider. This interface implementation is limited because the service implementation is tightly coupled with the current execution context [2].

But when other implementations want to use the same interface, they may fail because they may not know how to get the value of User ID or they may choose to acquire its value through a different way in that implementation. More over if different clients or assemblies want to use this implementation (Logging-in), each will have their own way of getting the User ID. This will result in different implementations for getting each User ID.

To prevent multiple implementations, it is advisable to implement the acquiring of the User ID as a parameter in the method signature itself as shown in the code below:

```
internal Interface ILogin{
        Func<int>LogPageAsync(Page p,string UserId);
        }
```

13.2.2 APIs as Systems exposing data and operations

In this type of API, there is data communication between different systems or services. This data exchange is performed through the internet. Hence the second type of API uses HTTP-based RESTful (Representational State Transfer) services with JSON (Java Script Object Notation) as a lighter data exchange format.

REST is fully featured and is based on a stateless, client-server, cacheable communications protocol. It uses the HTTP protocol to make remote procedure calls between machines. REST mechanism is simpler and more light weight when compared to other complex mechanisms like COBRA, RPC for remote procedure calls and SOAP, WSDL for Web Services.

RESTful applications use HTTP requests for all four CRUD (Create/Read/Update/Delete) operations. RESTful applications use the light weight JSON format for this. Code for reading and generating JSON data can be written in any programming language. Hence it is the more popular data format for data exchange on the Internet.

13.3 API ECONOMY

When APIs are sold for monetary benefit, it is termed as API economy. There are many instances where APIs are sold by Companies commercially. For example, ride sharing Applications like OLA and UBER use the Google Maps API in a pay-as-you-go model.

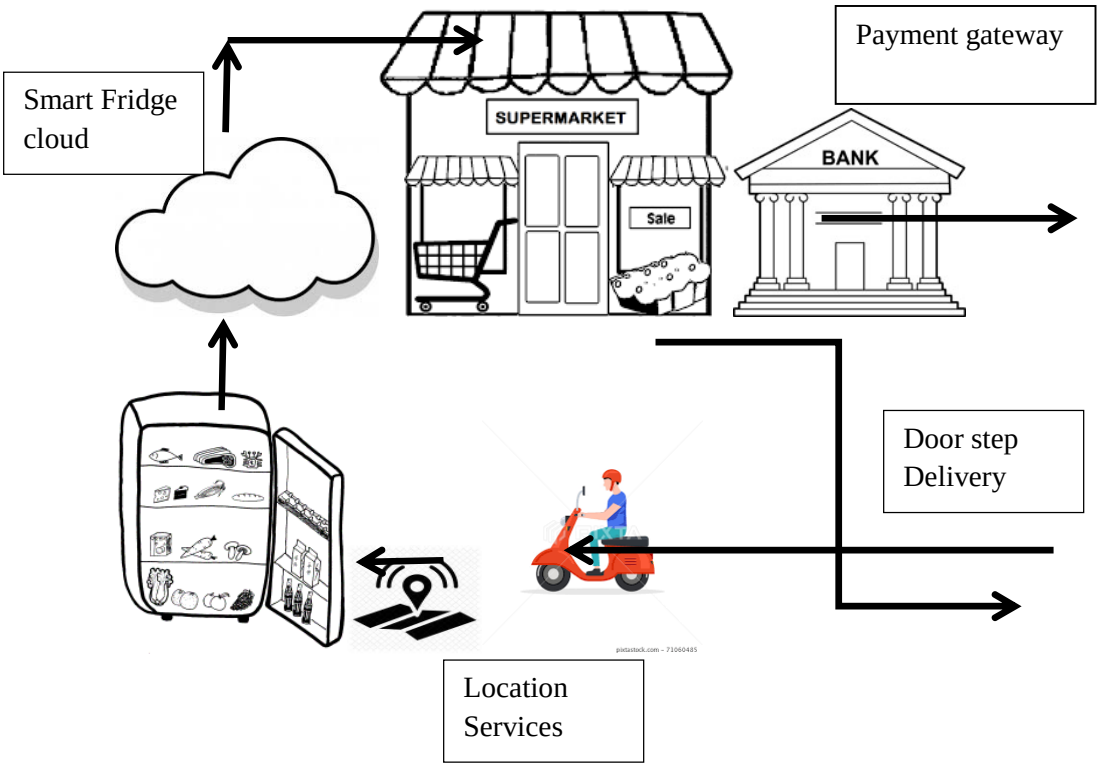
Businesses worldwide are competing with each other by developing new Applications using new technologies thus focusing on enhancing the user experience. These companies focus on focus on creating new customer experiences and find new opportunities to serve customers better and more efficiently.

A wearable device which will monitor a patient's vital signs can be convey metrics like heartbeat, pulse rate to hospital via the hospital's API. This in turn will ensure that the patient gets critical care in case of an emergency.

Food delivery applications use location-based searching allowing their customers to order food from their local eateries. The Food delivery application uses the APIs of Restaurants and other location-based searches available from other providers to create a new customer experience, all them benefit from this.

With innovations in the field of IOT and Sensors, one can think of a smart appliance like a smart fridge which can sense that a meat packet has expired and order for new packets through a mobile Application. When the meat reaches its expiry date the sensor in the refrigerator senses it and connects to the local grocery store API to reorder the meat. The meat is delivered to the customer's door step using location services like Google maps, as shown in figure 13.2

Fig 13.2 A smart fridge on cloud



Having a smart cloud and processing IoT sensor data will enable the fridge manufacturer to have a competitive edge over his competitors while giving his customers a value-added experience. The Supermarket accepts orders via its API through the Internet. The fridge manufacturer may even tie-up with multiple supermarkets charging them all in the process. The Supermarket accepts payment via the payment gateway API using a commission-based business model. Delivery can be done through other utility service APIs like location service to complete the delivery. These APIs can be either free or paid.

The above explains the different uses of different APIs for different business models. There APIs for direct selling, APIs offering their customers an improved customer experience and for creating an API ecosystem. Some businesses also offer their APIs on a commission model for E-commerce applications. All these types of API help to generate revenue for their businesses.

13.4 APIS IN PUBLIC SECTOR

APIs are used in public sectors to help governments to share their data and also to integrate their APIs with other public or private APIs.

Governments having been digitizing key government systems and procedures via e-governance. This has made governance more transparent and accessible to common man.

There are three government service models**[2]**.

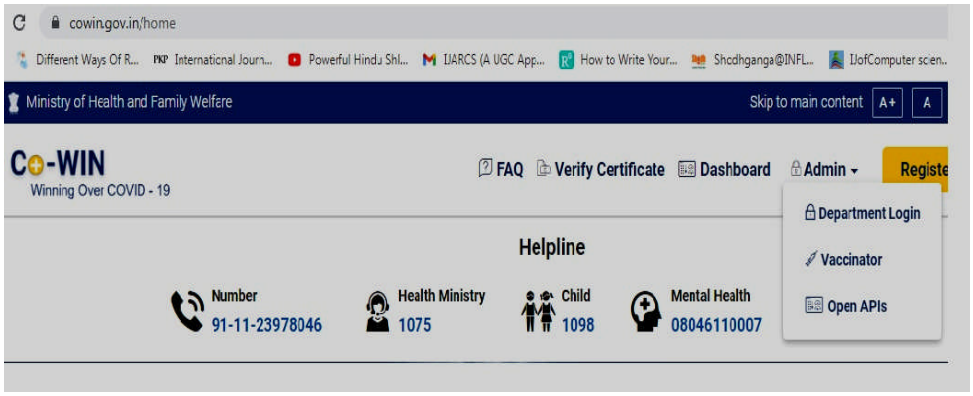
- •G2C: Government to Citizens
- •G2B: Government to Business
- •G2G: Government to Government

13.4.1 Government to Citizens

Governments all over the world are automating their services and allowing their citizens to access these services easily online. Government of India has given a unique identity to each and every citizen with the help of an Adhar card. Every Citizen can register himself/herself on https://uidai.gov.inportal and get an Adhar card. This portal is integrated with other Government portals like the Government of India, Income tax portal etc. and provides a whole some seamless digital experience to the Citizens of India

Government India also has an application called Dig Locker where Citizens can store all their electronic documents like their Adhar card, PAN card and their educational certificates digitally. Here the Government to Citizens model is being used.

The https://www.cowin.gov.in/home portal of Ministry of Health and Family welfare portal of Government of India has many open APIs like Public, Private and Vaccinator APIS for its users to register and get vaccinated in India for the Covid 19 Virus. This portal has digitized the process of user authentication, registering and getting vaccination slots anywhere in India for its citizens



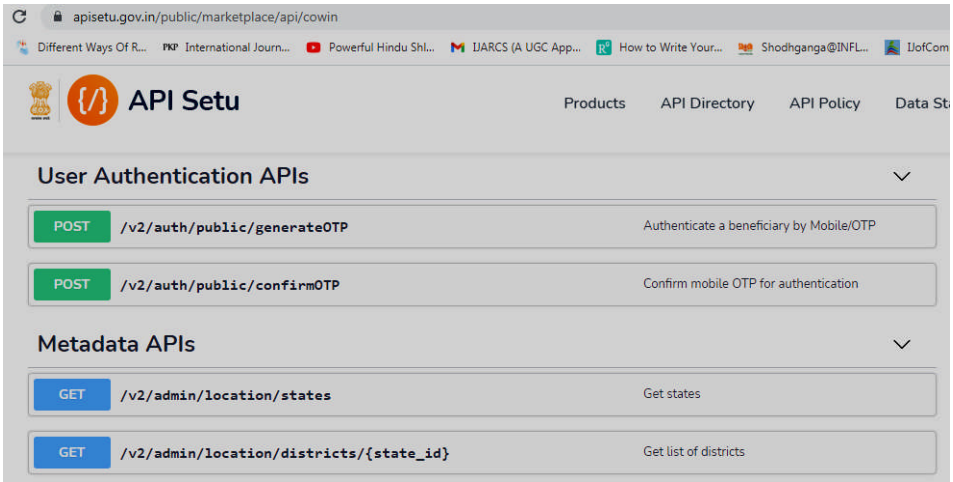


Fig 13.3 Open APIs on Cowin porrtal

13.4.2 Government to Business

Governments are also providing services to Businesses to simplify the ease of doing Business within India. A fine example is the portal https:/www.india.gov.in/india-business-portal. This is the National Portal of India, developed with an objective to enable a single window access to information and services being provided by the various Indian Government entities [3]. This Portal has been developed under the National E-Governance Plan, to simplify the ease of doing Business in India.

Another example is the use of Adhar card for identification and verification purpose for commercial purpose. When Customers apply for-Sim cards, they can be verified through their Adhar card on the https://uidai.gov.in government portal. Here the mobile service provider is using the government API or services for its business purpose.

Government of India's API setu portal provides Financial Verification APIs for Banks and Insurance companies in India. These API allow lenders use the financial data services to check credit worthiness of their customers.

Another example of government to businesses is the Singapore government's Smart Nation portal. This portal (https://www.smartnation.sg/resources/open-data), has numerous government APIs and data endpoints that serve citizens and businesses [2].

Digitization of government APIs improves efficiency, and brings new sources of revenue to governments. Also use of these APIs, gives businesses access to government's rules and regulations for establishing a businesses, government's taxation policies, business information, and regulatory audits etc.

13.4.3 Government to Government

Government to Government service models is when

- Various public sector departments interact with each other for sharing data or services or when state governments interact with the central / Union government APIs for data sharing or consume the services offered by the APIs of the central government.
- Other governments interact with the central / Union government APIs for consuming the services and for data sharing.

The **https://negd.gov.in/open-api** portal of government of India plans to implement a Policy Framework consisting of "Policy on Open Standards for e-Governance" in India. Implementation of this API Gateway shall provide an enabling platform for proactive and open access to the data generated by various ministries/departments/organizations of Government of India [4]. This portal is an example of the Government to Government service models integrating the various departments within a government.

13.5API Strategy

The importance of APIs for businesses and the monetary gains they bring has made the top most management get involved in the API strategy and architecture. As already discussed there are two types of API- private and public. The strategy to build as a private or public API is based on factors such as security, monetization strategy, data trends, and regulatory standards [2].

It is imperative for business stake holders to get involved in strategizing, creating so that they are aware of the data and operations API is exposing. They also should be able to set the expectations, goals, and constraints for the operational environment. The stake holders should evaluate different business models and work out an API strategy that will support their business vision.

The API strategy thus worked out should be implemented to ensure that the business goals are achieved. API architecture should integrate the technical aspects into the identified API strategy to realize the company's business vision.

The top management of a business sets the vision for the API and the technical team design and develop the API in order to realize the business goals.

The process of business decision making involving the planning, organization, or governance of an API is known as an API strategy [2].

To implement an API, it needs a lot of integration with other existing systems. The existing data flowing from legacy systems must be compatible across various domains used different API users.

For example, the Covid-19 pandemic and the ensuing lockdown saw a trend of online grocery shopping. Hence Supermarkets and retailers

have taken the decision to move their businesses online to ensure sustainability and gain profits. They should be able to use an API that will help them to expose their shopping lists online and attract customers. They have to integrate their existing offline system with an online system in a cost-effective way to ensure profits.

This means that they should start various development processes like:

- data cleansing,
- Using a common programming language
- Restructuring the source code of an existing application or legacy software so as to improve operation without altering its functionality.
- Updating the software tools and frameworks.

All the above steps are not part considered part of the API implementation, but form an important part of ensuring that the project's mission of improving the profitability of the business is accomplished.

13.5.1 API strategy with respect to strategic aspects of a business

The decision to implement APIs is undertaken taking into consideration business aspects like innovation, business operations, integration, and monetization [2].

The following are some of the strategic business aspects of an API implementation [2]:

1. **Business orientation:** An API implementation strategy is based on the business orientation of the organization and reveals the purpose of the business operations.

   For example:

   a. When the offline operation of a supermarket is integrated with an online operation, this will show the point of integration of the two systems. But in reality, it could be a **high-level business strategy** to two merge these two systems to increase the outreach of the business and to increase its profits.

   b. When travel websites offer flight booking and bus travel bookings along with hotel reservation, they do so **to create and develop business ecosystems**.

   c. When fulfilling a regulatory requirement, a health care service provider might expose certain trends in health data to government bodies. This could be a mandatory requirement in some countries.

2. **For technological innovation and changes:** Sometimes Organizations expose their data and operations through APIs to outsiders to attract new ideas, skills and innovations in their businesses. This will benefit the researchers and scientists who will get access to large amounts of data to conduct their research. Hence it becomes a win-win situation for both the parties.

3. **Monetary Purposes:** Organizations with valuable data and business operations sell them directly via APIs [2].

a. Organizations having valuable data or operations sell them as APIs, monetarily benefiting them. The users of these APIs pay some money to the Organizations. For example, Taxi aggregators pay Google to use its maps API.

b. To enable more business opportunities, Organizations may expose their operations as APIs. This helps in greater flexibility for integration and gives them the opportunity to be part of a business ecosystem.

Based on the above aspects of API business strategies it can be decided whether an API is private or public. This also decides whether the data and operations are to be exposed via APIs and its security and authentication. This also determines their venue generation strategies, usage policies, and restrictions.

When businesses share their data through public API it results in gain in revenue for the business and innovation in technology. Also, the developers can embrace other technologies to make the APIs adapt to latest technologies. However, when all data is publicly shared, it may lead to the competitors accessing sensitive business data and technology of the business. It will also raise security and privacy concerns. Hence the API strategy and architecture should determine, govern, and implement the correct exposure level of the correct data.

13.5.2 API strategy development stakeholders

To understand an API strategy development better, one need to consider who is involved and what capabilities of an API they want to use.

The following are the stakeholders involved in the API strategy development:

- Application developers: They consume the API. The application developer might work for the company that is exposing the API or work for a business partner of that company. Or, this person might not have any previous relationship with the organization that owns the API.
- API product manager: They person is responsible for defining various facets of the API, including the roadmap, target audience, monetization strategy, and lifecycle.
- API developer. They create the API and expose the IT assets of the organization.

13.5.3AN API Strategy Use Case

Government of India web site API Setu offers several types of APIs services like Adhar card, PAN card, driving license and Ration card to Banks, Insurance Companies etc. to build a strong risk profile of their customers.

Loan agents can connect to these services while offering loans to find the credit worthiness of the customers and then offer them loans. Before the government offered this citizen data to these financial institutions, they were collecting hard copies of the Adhar card, PAN card, driving license and Ration card etc. and manually verifying the data.

Under the vision of Digital India, the Government of India aims to make all the above Government services digitally accessible to citizens through multiple channels, such as web, mobile and common service delivery outlets.

The purpose of offering services is to expose APIs to integrate with banks and insurance companies, which are used to determine the credit worthiness of their customers. This creates an ecosystem of APIs which is employed to achieve smoother data flow.

The usage of 'Open APIs' is to promote software interoperability for all e-Governance applications & systems and provide access to data and services for promoting the participation of all stakeholders including citizens [5].

13.6 API VALUE CHAIN

An API implementation involves different levels and layers of an organization. Modern API implementations often include external stakeholders and other value providers like partners, suppliers, customers, and developers [2].

APIs integrate and digitize business flows by connecting different stakeholders with organizational IT assets [2]. API value chain refers to the entire ecosystem and the relationship between assets, API providers, and API consumers.

Supposing a garment company retailer wants to sell his products online. This decision was taken increase his outreach through online and mobile based applications. The inventory management system, product catalog system and the sales management systems have to be exposed via an API with minimum cost.

These existing software systems will interact with the API layer. And the API layer is utilized by a developer either for a web site or a mobile Application. The developers will then publish the apps that consume this API. Users of the apps then download and use them. Published apps will create an app ecosystem. The app users are unaware of the above API value chain. Figure 13.4 shows an API value chain.

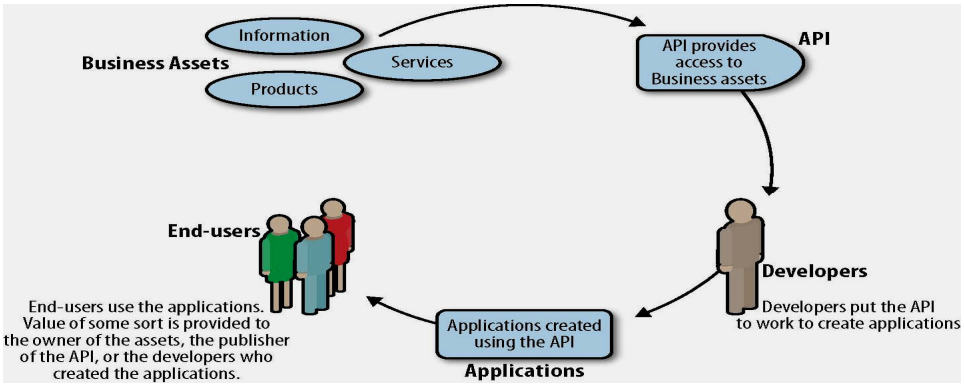


Fig 13.4
source:https://thenewtechnicalwriter.wordpress.com/2015/08/10/understanding-the-api-value-chain/

API should facilitate a developer community and deliver a proper developer experience in order to maintain steady engagement with the developers [2]. There will be two types of developers for these APIs. some developers are involved in the technical aspects like documentation, the on boarding process, and SDKs. The other types of developers will create content that will result in revenue generation for external developers. This type of content includes content such as commission strategy and advertising policies. Examples of this could be advertisements inserted in online games etc.

13.7 API ARCHITECTURE

APIs can be classified as two types depending on their capabilities and functions.

- System APIs. These APIs expose core back-end systems capabilities. They usually trigger generic, process-agnostic activities and tend to target application developers who are internal to the organization.
- Interaction APIs. These APIs support more use-case-specific functions. Their implementation often aggregates and filters calls to system APIs. The interfaces are optimized for ease of consumption and tend to target mobile devices or consumers who are outside the boundaries of the organization.

The purpose and strategy for developing an API is decided by the business stakeholders like Company CEOs, enterprise architects, data stewards, and other organizational evangelists based on the effort, budget, organizational context, and current model of IT assets needed to execute API implementation. The actual implantation of the API is done by the technical stakeholders. The implementation teams interacts with business stake holders to understand the purpose, execution, and limitations of the strategy.

For example, the CEO of a retail garment store chain has the devise ways of increasing the revenue for the company. Upon consulting his sales team, he finds that the online sales are steadily increasing. He then decides to create a mobile app for online retail selling to give an easy and comfortable interface for his customers.

He then along with the API strategy stakeholders initiates a strategy to develop an API. The technical stakeholders are responsible for creating and implementing the API. It is the job of the technical stakeholders to decide on the API architecture and identify technical aspects.

API architecture has six important attributes [2] to be fulfilled. Each of these attributes has its own risks and concerns which influences the overall API design in a given business context.

- Developer experience: The success of an API implementation depends on adoption by developers. Developers should be well versed with all aspects of an API design with the help of developer portal, forums, developer tools, and trail API endpoints etc.

- Integration: Many enterprise-grade API implementations are integrated with many systems. The majority of these systems are legacy systems. The API directly interacts with an existing legacy system or connects to them through a wrapper API.A common function of an API to first combine results from different systems and performing some data operations on them. Then the data is exposed through an API gateway. Some API gateways have the tools to do basic data translation and transformation capabilities.

- Performance: Responsiveness and availability are two performance requirements of an API. Caching is a used to increase the responsiveness of APIs. Many-chained integrations and slow data translations from legacy systems are responsible for performance degradation [2].

- Security: Security is a very important and critical attribute of an API. It should take care of user authentication, authorization. It should also prevent security risks like injection attacks, and DDoS attacks.
  In business context there will be security issues concerning data. What data and operations need to be exposed and how and whom to expose them are important things to be concerned about during API strategy. Exposing data and operations that reveal internal information about the business could create an advantage for a competitor.
- Usage & Telemetry: Measuring usage and logging helps in understanding the usage and adoption of the API. Endpoint-based analysis reveals patterns in how API endpoints are consumed and which endpoints are consumed together [2]. These details will help to continuously optimize the API design. There a lot of tools available in the API market for this.

- Error handling: Error handling determines the behavior of an while dealing with application-level errors and system failures. The application-level errors or failures are addressed by the API architecture. Application-level error handling should deal with error contracts, error documentation, error contract information level, security, and certain access limits. System failures are addressed by the system architecture.

13.8 API MANAGEMENT

API Management refers to a collection of tools used to design and manage APIs. It also refers to both the standards and the tools used to implement API architecture. Some of the API Management product vendors are enterprise tools like Microsoft, AWS and IB and vendors like Apigee, Sales force, and WSO2.

All these management tools offer the following solutions [2]:
API design: API design offers functionalities like importing an API from specifications, creating API endpoints, defining service contracts and generating documentation.

API Gateway: An API gateway engine is used for manipulating requests and responses, URL rewriting, caching, security enforcement and pre-authentication and applying request-based security rules.

API analytics: API analytics which is usually configured as part of the API gateway gives analysis of the API usage. API analytics is tracked and monitored and provide usage, telemetry insights and reporting dashboards.

API catalog: An API catalog in vendor-specific. But a catalog usually lists the available APIs and other access configurations. For example, one API Management solution can have many APIs; some of these APIs may be public, while the others could be private. Access to private API will be usually associated with a certain authentication.

Further features like API documentation, API publishing, protocol translation, data translations, data transformation, security capabilities, developer portals, caching, versioning, client SDK generation, usage and telemetry monitoring, URL rewriting etc. [2] are all part of the above API Management tools.

API Management and the tools have a huge market in the IT industry. Many integration service providers offer API Management tools, and enterprise usage of API triggers the demand for API Management solutions [2].

SUMMARY

This chapter focused on introducing the basics of API to the learner. It also walks them through concepts related to a practical implementation of APIs based on language constructs and as Systems exposing data and operations. It also introduced concepts related to API in economy and public sector. It also takes the learner through API value chain which refers to the entire ecosystem and the relationship between the assets, API providers, and API consumers.

It introduces the six important attributes of API architecture. Some of the API Management product vendors are enterprise tools like Microsoft, AWS and IB and vendors like Apigee, Sales force, and WSO2 and the solutions they offered are mentioned.

REFERENCES:

1.**https://www.wrike.com/blog/application-programming-interface-api-explained/#:~:text=Other%20examples%20of%20APIs%20that,constellation%20data%20for%20public%20use.** *By Andrew Slate, May 31, 2019*

2. Practical API Architecture and Development with Azure and AWS-Thurupathan Vijayakuma ,*ISBN-13 (pbk): 978-1-4842-3554-6,ISBN-13 (electronic): 978-1-4842-3555-3,https://doi.org/10.1007/978-1-4842-3555-3 ,Library of Congress Control Number: 2018946567*
3. https://www.india.gov.in/india-business-portal
4. **https://negd.gov.in/open-api#features**
5. **https://apisetu.gov.in/api-policy.php**

MODEL QUESTIONS

MULTIPLE CHOICE QUESTIONS

1. _____save the developer's time by allowing them to use previously existing tools within their applications, thus reducing the time and cost for creating these tools.
   A) External APIs B) internal APIs C) Database D) Search Engine

2. The API standards are also called_____.
   A) Frameworks B) web service protocols C) Gateways
   D) Search Engine

3. REST, which stands for _____
   A)Representational state transfer B) Resource efficient state transferC) Reverse state transfer D) none

4. When APIs are sold for monetary benefit, it is termed as API _____.
A) Gateway B) Economy C) E-Commerce D) management

5. There are _____ number of government service models for APIs in public sector.
A) two B) threeC) one D) four

6. The _____verb is used to get a single resource or list of resources.
A) GET B) SET C) PUT D) DELETE

7. _____ is an important API Development Considerations.
A) Frameworks B)Using Explicit Parameters C) Gateways
D) Search Engine

8. _____ is a way to handle API versioning standards.
A) View state B) Cookie C)Query string parameter
D) Session variable

9. _____is an Interface Description Language for describing RESTful APIs,.
A) HTML B) XML C) Swagger D) Mango DB

10. _____ is an API implementation standard.
A) HTTP status codes B) XML Specification C) Document Type Definition D) none

Answers to Multiple Choice Questions
1. A 2.B 3.A 4.B 5.B 6.A 7.B 8.C 9.C 10.A

Theory Questions
1. Explain the government service models for APIs in public sector?
2. Explain the strategic business aspects of an API implementation?
3. Who are the stakeholders in API strategy development?
4. Explain the API value chain?
5. State and explain the attributes of API architecture?
6. Explain API management?

14

API DEVELOPMENT

Unit Structure :

14.0 OBJECTIVE

This objective of this chapter is to give a basic understanding of the standards involved in implementing APIs. It discusses details of the API implementation standards such as URI, HTTP verbs, exceptions, developer experience, HTTP status codes and naming conventions.

14.1 API DEVELOPMENT CONSIDERATIONS

The purpose of an API is to lets the components of two applications talk to each other using a set of simple commands. Essentially, APIs are messengers that deliver requests and return responses between applications. They are different from web applications and web services. APIs are similar to a RESTful service implementation. But an API has more capabilities than RESTful services. APIs with RESTful semantics can be considered to be RESTful services. But not all RESTful services are APIs.

Usability and adaption of an API by various external callers are important considerations which are critical to the success of an API implementation. In order to develop a good API which can evolve over time the following considerations need to be kept in mind:[1]:

1. Using Explicit Parameters

API implementations should receive parameters from external callers and should not rely on any client-side, state-persistent models. Generally when a RESTful service is developed for a specific web application where a single client will interact with the server, these services will accept data from client-side cookies. But if this RESTful service is used to serve as an API for several consumers then accepting data from cookies is not acceptable as it creates problems.

Hence API implementations should have explicit parameters and accept data from URL parameters or HTTP headers or via HTTP request body[1], instead of relying on cookies

2. Avoiding Consumer-Commanded Endpoints

APIs and their consumers send and receive messages using defined service calls. Service calls can be defined by the API service or it can be-defined by the API consumers. Both methods are correct. But the problem with consumer-defined calls is that they are application specific. API service calls on the other hand are based on entities and business operations and are common to all applications.

However, it is best to avoid endpoints that serve consumer-commanded data. Consumer commanded data is the service call definitions which contain the specific application's view model, Examples of such service calls are various formatting of data, APIs exposing endpoints for simple data aggregations and API responses containing visual styles like color codes. This kind of service calls tightly couple the API implementation to a specific client and a specific application view and hence should be avoided.

3 .Creating extensive documentation

While developing an APIs there should be extensive documentation about the used standards, version, URI syntax, and error codes [2]. This kind of documentation helps other developers to understand and implement this API. To create documentation, tools like Swaggerand TRexcan be used. Full-fledged API Management tools provide rich documentation and developer experience [2].

4. Incorporating Security

API security should not only take care of user authentication and authorization, but also deal with data that is to be exposed via a service call. It should also deal with how endpoints are consumed by the consumers.

Usually when there is an error while sending a request, there is a response which is returned with the appropriate error code and description of the error containing a data property. This may help in rectifying the error by resending the request with the correct request parameters. But at the same time it will create a security loop hole. This loophole can be exploited by a hacker to obtain some internal sensitive data. Hence it important to have APIs having helpful responses to clients while not exposing sensitive information. Public APIs should also implement security measures such as IP-based security, tracking the usage of the API key, or limiting the call rate. Modern API Management tools offer these request based security methods.

5. Creating further versions

APIs are software, and software evolves. API development should consider the versioning; versioning of APIs cover two aspects, one is the versioning of the URI and second is the versioning of the service contract. There are many API versioning techniques available, and the proper technique should be chosen in the early stages of development. Prompt notification to developers about new versions and especially the depreciation of old versions are essential.

14.2 STANDARDS

This section explains the best standards to be followed while using some key elements in HTTP communication like HTTP verbs, and status codes. It also describes the standards to be followed for doing error handling, the URI syntax standards and the API versioning standards.

14.2.1 HTTP verbs

APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. These are the methods used for HTTP communication between a client and a server.

Table 14.1lists the important HTTP verbs that are most commonly used in API development. These methods are used to create, retrieve, update, delete entities or resources through a client Request method sent to the server.

Table 14.1HTTP verbs

GET	used to get a single resource or list of resources	Read	HTTP GET http://www.appdomain.com/users?size=20&page=5
POST	used to create an resource	Create	HTTP POST http://www.appdomain.com/users/123/accounts

PUT	Replace an resource	Update/ Replace	HTTP PUT http://www.appdomain.com/users/123
PATCH	Update the properties of an resource	Update /Modify	HTTP PATCH /users/1 [{ “op”: “replace”, “path”: “/email”, “value”: “new.email@example.org” }]
DELETE	Delete an resource	Delete	HTTP DELETE http://www.appdomain.com/users/123

Each of the above verbs has specific functions. PUT and PATCH methods even though appear to do the same, there is a basic difference between them both. The PATCH method was introduced to do partial resource modification. The PATCH request contains a set of instructions to modify an existing resource on the server to produce a new version. The PATCH method affects the resource identified by the Request-URI, and it may also affect other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH. The PUT request however is used to replace an existing version o the resource on the server with the new version.

14.2.2 HTTP status codes

HTTP status codes indicate the state of a response from the server, got through the HTTP response object. Some API implementations have their own HTTP status. When a client makes a request to an HTTP server and the server successfully receives the request, the server must notify the client if the request was successfully handled or not through status codes**.** Table 14.2 shows some common HTTP status codes.

Table 14.2 HTTP status codes[3]

Status Code No.	Status of the Request	Description of the status
200	OK	The request was successfully completed.
201	Created	A new resource was successfully created.
400	Bad Request	The request was invalid.
401	Unauthorized	The request did not include an authentication token or the authentication token was expired.
403	Forbidden	The client did not have permission to access the requested resource.
404	Not Found	The requested resource was not found.
405	Method Not Allowed	The HTTP method in the request was not supported by the resource. For example, the DELETE method cannot be used with the Agent API.
409	Conflict	The request could not be completed due to a conflict. For example, POST Content Store

		Folder API cannot complete if the given file or folder name already exists in the parent location.
500	Internal Server Error	The request was not completed due to an internal error on the server side.
503	Service Unavailable	The server was unavailable.

It is recommended to return a BAD Request error rather than returning a Not Found error for the status code 404since the original resource is there one the server, but the client has sent a wrong resource name.

Some API developers use their own status codes. But it is a recommended best practice to use the above response status codes with descriptive error messages in the body especially when a status codes like 400 or 500 are returned.

14.2.3 Error handling

Error handling is an important way in dealing with incorrect Requests. The best practice followed in API development while handling errors is by returning an error response with at least three parameters. The following three parameters should be usually returned in response to an error [2]:

1. Correct HTTP status code like status code number
2. API-specific error code
3. Human-friendly error message

Sample response message

```
"timestamp":"2019-09-16T22:14:45.624+0000",
   "status":500,
   "error":"Internal Server Error",
   "message":"No message available",
   "path":"/api/book/1"
```

API-specific error codes help in implementing the client logic easily, rather than processing the human-friendly string message. This also helps in implementing good flow control logic in clients [1]. The error response can contain details such as retry links, retry time interval, and additional helping parameters to modify the response object. Table 14.3 gives the problem detail response object properties as defined in the RFC 7807 document. The response object can be extended with custom properties.

Table 14.3 Error handling- Problem Detail Message as Specified in RFC 7807

Type	String	URI for the type of the error
Title	String	Short description of the error
Detail	String	Detailed description of the error
instance	String	Instance of the error

The following is a Sample Problem Detail Message as Specified in RFC 7807

```
{
"type": "https://example.com/probs/out-of-credit",
"title": "You do not have enough credit.",
"detail": "Your current balance is 30, but that costs 50.",
"instance": "/account/12345/msgs/abc","balance": 30
}
```

For example sending the current balance in the error response is an important decision which should be weighed carefully while considering its advantages and disadvantages. If the overall security is compromised by these kinds of messages then the developer should refrain from returning them.

14.2.4 URI Syntax

A request URI in a HTTP-based RESTful service sends parameters in the request URI or HTTP headers, or in the request body. API developers decide what parameters are sent on which path. For simple parameter passing a GET request method can be used to pass the parameters in URI. But if there is a specific need to send complex search parameters then they are sent in the request body rather than in the URI.

URI syntax can be either query string based or URI fragment based. Modern API frameworks support both syntaxes. A query string based URI syntax works fine if the specific naming convention like using a "?" followed by name value pairs is used. URI fragment based syntax is generally written as –"api/orders/1" .This syntax is preferred due to its general semantic based approach [1]. Table 14.4 shows the syntax for both query string based URI and URI fragment.

Table 14.4

api/orders?id=1	query string based URI syntax
api/orders/1	URI fragment

To make the URI syntax more readable one can also write the parameters in the middle as shown in Table 14.5

Table 14.5

api /order/**1**/products	URI fragment with parameter **in the middle** as shown in bold in the syntax
api / order / products? **order Id = 1**	URI fragment with parameter`as shown in bold in the syntax

URI fragments are not manageable in all scenarios. When used with search/filter operations requiring a search endpoint with arbitrary parameters the URI fragments approach is not convenient.

For example if we are require a filter operation which takes two parameters values color and size, we can introduce a search filter in the following manner

api / products / filter / { color } / {size}

Here there are two parameters, one is the filter's key parameter and second property is the value of the parameter. In above example, color and size are the two parameters which can take some value like "red" and "medium". But chaining a lot of parameters in a URI is restricted by the URI length limitation.

When complex search operations are involved it is better to send a GET request with the payload that defines properties, values, and filter criteria in the request body. This can be accomplished as shown in the code below [2]:

```
var request = new Http Request Message
{
   Method = Http Method.Get,
   RequestUri = new Uri("https://localhost:44341/api/auth"),
    Content = new String Content (json, Encoding.UTF8, "application/json")
};
```

Using the above URI segmented approach allows the URIs to be more human friendly, and keeps the request URIs clean from characters like "?", "=", and "#"[1].

14.2.5Versioning

API versioning standards are handled in the following three ways:

Table 14.6

1.	As a API version in the URL as a fragment (common method)	http://api.example.com/v1

2.	As a query string parameter:	http://api.example.com/ / products? api-version=1
3.	As a custom HTTP header (e.g. Accept-version)	Accept-version: v1

The easiest and simplest way of sending the version is having the version in the URL as shown in Table 14.6.Whichever method is used, APIs expect the version from the client as a parameter, and if the client does not specify the version, the API either falls back to a default version or throws an error [1].

The header key "api-version" is used to pass the version to the server as shown in Table 14.6 in row 2.But developers can use their own custom header keys or state the API version in the Accept header key. In the third method in row3 of Table 14.6 a custom Header – "Accept-version" was used to send the version information.

To state the **API version in the Accept header key** the following method can be used:

```
Accept: application/vnd.example+json;version=1.0
```

The above header is used for content negotiation in a JSON-based API with the version number at the last. Some APIs expect the version number to be placed before "json" in the Header as follows:

```
Accept: application/vnd.example.v1+json
```

Regardless of the method used, implementation should be consistent across all endpoints of the API. Also, one API can have more than one method enabled in its implementation.

14.3 KICK-START API DEVELOPMENT

While discussing API development in this chapter the OAS specification is used along with the Swagger tools for implementing it. The next section describes the tools and specification used for API development.

14.3.1 Swagger tools and the Open API Specification for API development

Swagger is an Interface Description Language for describing RESTful APIs, expressed using JSON.

Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services. It includes both API specification and tools.

The current version of swagger is called the Open API Specification (OAS) specification.

Open API Specification (OAS) is the specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services. It allows both humans and computers to discover and understand the capabilities of the service without access to its source code.

The OAS specification has been made vendor neutral but the implementation tools still go by the brand name- Swagger. While discussing API development in this chapter the OAS specification is used along with the Swagger tools for implementing it.

14.3.2 The ASP.NET Core implementation for a sample API

In this chapter theASP.NET Core which is the open-source version of ASP.NET is used for developing a sample API called "my Sample API". To create a simple ASP.NET Core Web API in Visual Studio use the following steps:

- From the File menu, select **New > Project**.
- Select the **ASP.NET Core Web API template** and click **Next.**
- Name the project my Sample API and click Create.
- In the Create a new ASP.NET Core Web Application dialog, confirm that .NET Core and ASP.NET Core 5.0 are selected. Select the ASP.NET Core Web API template and click Create as shown in figure 14.1

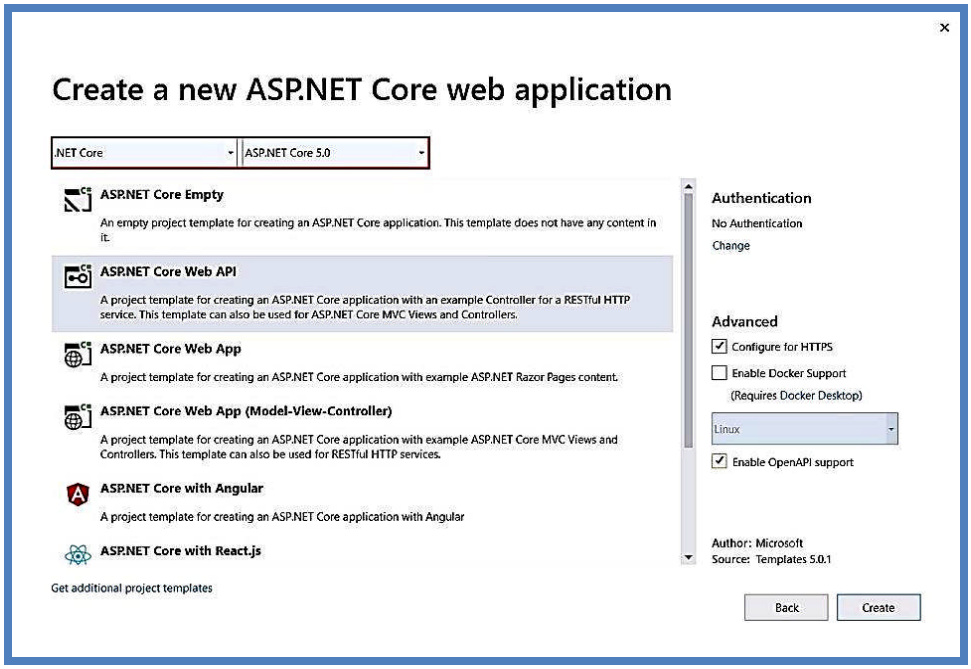


Figure 14.1 Visual Studio dialog box for creating a ASP.NET Core web API Project

In the next step, create a **new folder** in the above project and name it "Models." **Create the Product Model "*Product.cs*" in this folder.**

Product.cs

```
public class Product
        {
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime? ModifiedDate { get; set; }
        }
```

Next add another **new folder "Errors."** This folder will contain all the used classes .It will also contain enums to provide error handling implementation of our sample API as defined in the RFC 7807.

Create the following "Error.cs " file in the "Errors" folder

ErrorCode.cs

```
public enumErrorCode
{
RequestContentMismatch = 18000,
EntityNotFound = 18500
}
```

The sample "ErrorCode" enumis not implemented according to any general API standards. Its implementation is specific to this application.

One has to keep in mind to make sure that the error codes are consistent across the application and documentation, as API consumers should make decisions and write code based on defined standards.

Now create an abstract class in the Errors folder, named "ErrorMessage.cs," with the base properties of the error contract, and implement two specific Error Message concrete classes named "Request Content Error Message" and "Entity Not Found Error Message."

ErrorMessage.cs

```
public abstract class ErrorMessage
{
public ErrorCode Code { get;set; }
public string Type { get; set; }
public string Title { get; set; }
public string Detail { get; set; }
public string Instance { get; set; }
public string Info { get; set; }
}
```

```
public class RequestContentErrorMessage : ErrorMessage
{
        public RequestContentErrorMessage()
        {
        Code = ErrorCode.RequestContentMismatch;
        Type = $"https://massrover.com/doc/errors/#
        {ErrorCode.RequestContentMismatch.ToString()}";
        }
}
public class EntityNotFoundErrorMessage : ErrorMessage
{
        public EntityNotFoundErrorMessage()
        {
        Code = ErrorCode.RequestContentMismatch;
        Type = $https://massrover.com/doc/
        errors/#{ErrorCode.EntityNotFound.ToString()};
        }
}
```

Next create another class Error Service to return the correct Error Message instance in the right context. This class simulates a service that produces the correct Error Message instance. In a real-world implementation, this logic should be made part of the business logic.

ErrorService.cs

```
public static class ErrorService
{
        public static ErrorMessage
GetRequestContentMismatchErrorMessage()
                {
                   return new RequestContentErrorMessage
                   {
                     Title = $"Request content mismatch",
                      Detail = $"Error in the request context."
                    };
                }
                    {
                   ErrorCode.RequestContentMismatch.ToString()}";
                      }
    }
public class EntityNotFoundErrorMessage : ErrorMessage
{
```

```
        public EntityNotFoundErrorMessage()
        {
        Code = ErrorCode.RequestContentMismatch;
        Type = $https://massrover.com/doc/
        errors/#{ErrorCode.EntityNotFound.ToString()};
        }
}
public static ErrorMessageGetEntityNotFoundErrorMessage(Type entity,
int id)
{
  return new EntityNotFoundErrorMessage
  {
   Title = $"{entity.Name} not found",
   Detail = $"No {entity.Name.ToLower()} found for
    the supplied id - {id}"
    };
}
}
```

In the next step, create an ASP.NET Core controller with the actions for the CRUD operations of Product entity. In order to do this, add an empty Web API controller named "Products Controller" with the endpoints as listed in Table 14.7.

Table 14.7*ProductsController Endpoints*

Action Name	HTTP Method	Response Error	Response
GetProducts	GET	200 List of Products	-
GetProductById	GET	200 Products	404 - Entity Not Found
CreateProduct	POST	201 New Product	-
UpdateProduct	PUT	204 No Content	400 - Bad Request 404 - Entity Not Found
DeleteProduct	DELETE	204 No Content	404 - Entity Not Found

.

To develop the code for Product Controller.cs, first add a product collection in the controller as shown in the code below:

```
[Produces ("application/json")]
[Route ("api/products")]

public class Products Controller : Controller
{
private static List<Product> _products = new
```

```
List<Product>
{
new Product {Id = 1, Name = "Lithim L2",
Modified Date = DateTime.UtcNow.AddDays(-2)},
new Product {Id = 2, Name = "SNU 61" }
};
}
```

Next, add two HTTP GET actions. “Get Products”, GET action will be to retrieve all the products, and “Get Product ById” to retrieve a product by its ID. Parameters for the above HTTP GET action are passed via URI path. The Action methods like the GET, POST have proper attributes for the HTTP method, route parameters, and response types. Each action type has its own XML comments.

```
/// <summary>
/// Gets list of all Products
/// </summary>
/// <returns>List of Products</returns>
/// <response code="200">List of Products</response>
[HttpGet]
[Produces Response Type (type of (List<Product>), 200)]
public IAction Result Get Products()
{
return Ok(_products);
}
/// <summary>
/// Gets product by id
/// </summary>
/// <param name="id">Product id</param>
/// <returns>Product</returns>
/// <response code="200">Product</response>
/// <response code="404">No Product found for the
specified id</response>
[HttpGet("{id}")]
[ProducesResponseType(typeof(Product), 200)]
[ProducesResponseType(typeof(EntityNotFoundError
Message), 404)]
public IActionResultGetProductById(int id)
{
var product = _products.SingleOrDefault(p =>p.Id
== id);
if (product != null)
return Ok(product);
else
return NotFound
(ErrorService.GetEntityNotFoundErrorMessage
(typeof(Product), id));
}
```

Next create an actionHTTP POST to create new products

```
/// <summary>
/// Creates new product
/// </summary>
/// <param name="product">New Product</param>
/// <returns>Product</returns>
/// <response code="201">Created Product for the
request</response>
[HttpPost]
[ProducesResponseType(typeof(Product), 201)]
public IActionResultCreateProduct([FromBody]Product
product)
{
product.Id = _products.Count + 1;
_products.Add(product);
return CreatedAtRoute(new { id = product.Id }, product);
}
```

Add the PUT method for replacing products with the specified ID. This action requires two parameters:

The ID of the product to be replaced (this is passed as a path variable) and the product to be replaced with the new values, which is passed in the request body.

This action returns a NoContent response with the HTTP status code 204 for the successful replacement of the product. Otherwise, it responds with two different error contracts. One is “400 Bad Request,” with the request content mismatched when the ID value in the path does not matchthe ID value of the product in the request body. The other is “404 Not Found,” when the requested entity with the specified ID is not found.

```
/// <summary>
/// Replaces a product
/// </summary>
/// <param name="id">New version of the Product</param>
/// <param name="product">New version of the
Product</param>
/// <returns></returns>
/// <response code="204">No Content</response>
/// <response code="400">Request mismatch</response>
/// <response code="404">No Product found for the
specified id</response>
[HttpPut("{id}")]
[ProducesResponseType(204)]
[ProducesResponseType(typeof(RequestContentError
Message),400)]
[ProducesResponseType(typeof(EntityNotFoundError
Message), 404)]
public IActionResultUpdateProduct(int id, [FromBody]
```

```
Product product)
{
if (id != product.Id)
return BadRequest(ErrorService.GetRequest
ContentMismatchErrorMessage());
var existingProduct = _products.SingleOrDefault
(p =>p.Id == product.Id);
if (existingProduct != null)
{
existingProduct = product;
existingProduct.ModifiedDate = DateTime.UtcNow;
}
else
return NotFound
(ErrorService.GetEntityNotFoundErrorMessage
(typeof(Product), product.Id));
return NoContent();
}
```

Create a delete endpoint using the HTTP DELETE action.

```
/// <summary>
/// Deletes a product
/// </summary>
/// <param name="id">Product id</param>
/// <returns></returns>
/// <response code="204">No Content</response>
/// <response code="404">No Product found for the specified
id</response>
[HttpDelete("{id}")]
[ProducesResponseType(204)]
[ProducesResponseType(typeof(EntityNotFoundError Message), 404)]
public IActionResultDeleteProduct(int id)
{
var product = _products.
SingleOrDefault(p =>p.Id
== id);
if (product != null)
_products.Remove(product);
else
return NotFound
(ErrorService.GetEntityNotFoundErrorMessage
(typeof(Product), id));
return NoContent();
}
```

The next major step in this API development is to set up theSwagger tools in ASP.NET Core and supply these tools with the required attribute elements.

14.3.2.1Setting-Up Swagger

1. To setup Swagger follow the below steps:
   First install the Swashbuckle. Asp Net Core package in the project by executing the following command in the Package Manager Console (PMC):

```
Install-Package Swashbuckle.AspNetCore
```

2. In the next step, set up the Startup.cs to activate Swagger tooling and get the Swagger UI up and running in the project.
3. Next update the Configure Services method in the Startup.cs, file, [1]as shown below:

```
public void Configure Services (I Service Collection services)
{
services.AddMvc();
services.AddSwaggerGen(c =>
{
c.SwaggerDoc("v1", new Info { Title = "MyAPI", Version = "v1" });
c.IncludeXmlComments
(Path.Combine (Platform Services.Default.
Application.ApplicationBasePath,
"MyAPI.QuickStartSample.xml"));
});
}
```

4. Update the name of the API (MyAPI) and the versionof the API (v1) in the SwaggerDoc, and update the path for the XMLdocumentation for Swagger to use via IncludeXmlComments.

5. In the “MyAPI.Quick Start Sample.xml” sample, install the package Microsoft.Extensions.Platform Abstractions using PMC in order to provide the XML path.

6. Execute the following commands:

```
Install-Package Microsoft.Extensions.PlatformAbstractions
```

7. Next update the Configure method as shown in the code below [1]. This will enable the Swagger UI and sets the endpoint of the Swagger definition.

```
public void Configure (IApplicationBuilder app,
IHosting Environment env)
{
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}
app.UseSwagger();
app.UseSwaggerUI(s =>
{
s.SwaggerEndpoint("/swagger/v1/swagger.json",
```

```
"MassRover Open API");
});
app.UseMvc();
}
```

8. Instruct Visual Studio to generate XML documentation based on the comments

For completing the documentation navigate to Project Properties, then Build tab, and enable the XML documentation file.

14.3.2.2Run the API and Swagger

With the above installation complete, run the application. Navigate to the **URL http://{host}:{specified port}/swagger**. Swagger's UI uses the XML documentation generated by Visual Studio and provides details of the endpoints, parameters, response types, and response codes. Swagger also provides a detailed description of the above components as described in the XML comments.

14.4 PUTTING TOGETHER A TEAM FOR API IMPLEMENTATION

API implementation strategy involves both technical skills as well as knowledge about the business environment. Multiple teams work for an API implementation. These teams have to work together to integrate different endpoints under a single API standard. This kind of team work is commonly used during a commonly utilized in micro services-based architecture implementations.

For micro services-based architecture implementations, different teams work on different services, thus creating APIs with different standards. But for a client accessing these micro services, there has to be a single standardized API experience. API teams play a key role in standardizing multiple streams of services under one standard channel of API [1].

SUMMARY

This chapter discusses certain key point developers have to keep in mind while doing API implantation. It also focuses on API implementation standards such as the following:

- URI format
- HTTP verbs
- Error handling
- HTTP status codes
- Versioning.

This chapter also talks about how to put a team together and start the process of API development using ASP.NET Core and related Visual Stu-

dio tools. It also shows how to produce documentation for an API specification describing a RESTful API. Open API Specification (OAS) along with certain other Swagger tools is used for this purpose.

REFERENCES

1. Practical API Architecture and Development with Azure and AWS-Thurupathan Vijayakuma ,*ISBN-13 (pbk): 978-1-4842-3554-6,ISBN-13 (electronic): 978-1-4842-3555-3,https://doi.org/10.1007/978-1-4842-3555-3 ,Library of Congress Control Number: 2018946567*

2.https://documentation.commvault.com/commvault/v11/article?p=45599.htm

3.https://documentation.commvault.com/commvault/v11/article?p=45599.htm

MODEL QUESTIONS

Multiple Choice Questions

1. When RESTful service is used to serve as an API for several consumers then it should accept _____.
   A) Explicit Parameters B) caching C) cookies D) none

2. While developing an APIs there should be extensive documentation about the _____..
   A) URI syntax B)Using Explicit Parameters C) Query string parameter D) none

3.The _____ HTTP Verb is used to create an resource.
   A) GET B) PUTC) POST D) none

4. The status code _____ indicates that the request was successfully completed.
   A) 404 B) 400 C) 200 D) 234

5. _____is an important way in dealing with incorrect Requests..
   A) Error handling B) bandwidthC) Throttling D) none

Answers to Multiple Choice Questions

1. A 2.A 3.C 4.C 5.A

Theory Question

1. State and explain some of the most considerations for developing a good API?
2. Write a short note on API standards?
3. Write a short note on Swagger tools and the OpenAPI Specification for API development?

❖❖❖❖

15

API GATEWAYS & API Security

Unit Structure :

15.0 OBJECTIVE

This chapter deals with API Gateways in public cloud. It discusses about how to manage the endpoint in an API. It then talks about two such

popular API management services in public cloud-Microsoft Azure and the Amazon based AWS. API Management and API Gateway are the respective brand names of the API management services from Azure and Amazon's AWS. One should not confuse these names with the generic terms API gateway and management.

It then discusses how security can be handled on a public cloud through various security mechanisms like Request-based security and authentication and authorization.

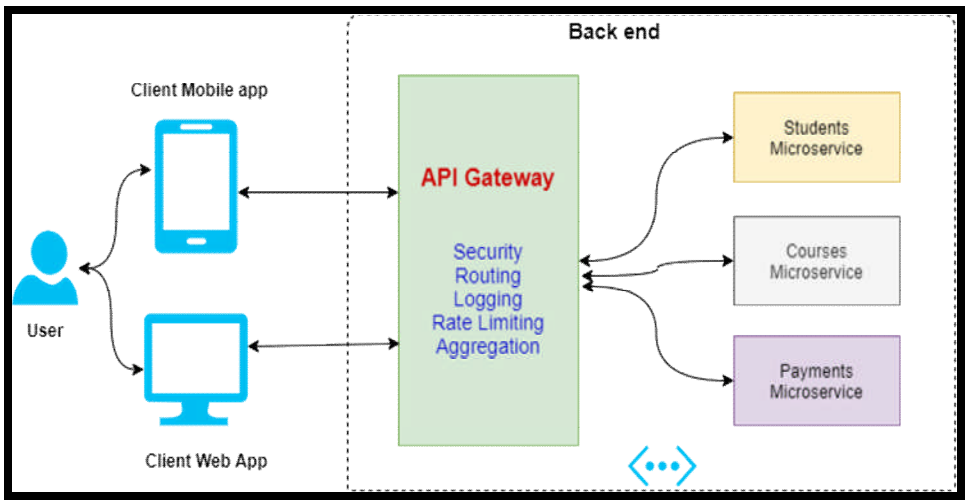
15.1 API GATEWAYS IN PUBLIC CLOUD

API gateways are like the CDNs which act as the front layer for APIs. Their main function is to abstract the underlying API or service and provide a uniform access point to consumers. They are the single point of entry for the various Micro services offered by the API.

An API gateway accepts all API calls and then acts as a reverse proxy, retrieving resources from backend applications on behalf of the client application. An API gateway not only accepts API calls — it also handles tasks related to API services like user authentication, rate limiting, monitoring, and more. A client submits its request and the gateway can break it down into multiple requests, route them to different backend services, and return the response to the client in a single round-trip. An API gateway therefore reduces the number of requests between the client and server, improving latency and the user experience.

When an API consumer sends HTTP/HTTPS request to an API service in a public cloud it is first received by the API gateway. The API Gateway then performs certain actions on the inbound request and passes the request to the backend service. It then performs actions on the outbound response. Figure 15.1 shows the scenario for an API Gateway for Student Management. In Figure 15.1, consumers like mobile applications, websites, IoT devices, other API services, and direct human consumers make a request to a public API Gateway through the internet. This is then passed on to the backend services after the gateway performs certain action. The outbound Response from the backend is passed on to the gateway for performing appropriate actions. The gateway then sends the Response to the consumer of the API. In the figure below the backend services are transparent to the end user. The API gateway has full control over the requests it receives, and has the same control over the response it delivers to the consumer.

Figure 15.1 An API Gateway for Student Management https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/



APIs also perform other value-added functions like caching, security, management, content negotiation, and policy management. A collection of these functions or services along with architecture components such as developer experience, enterprise integration, telemetry, access and request policies, and access control etc. forms the core of API management. Most commercial API gateways are offered as a part of the API management tools and services.

Figure 15.1shows a generic overview of a public cloud API. Internals of implementation, configuring APIs, connecting backend services, and configuring rules are vendor-specific.

15.2 ENDPOINT MAPPINGS

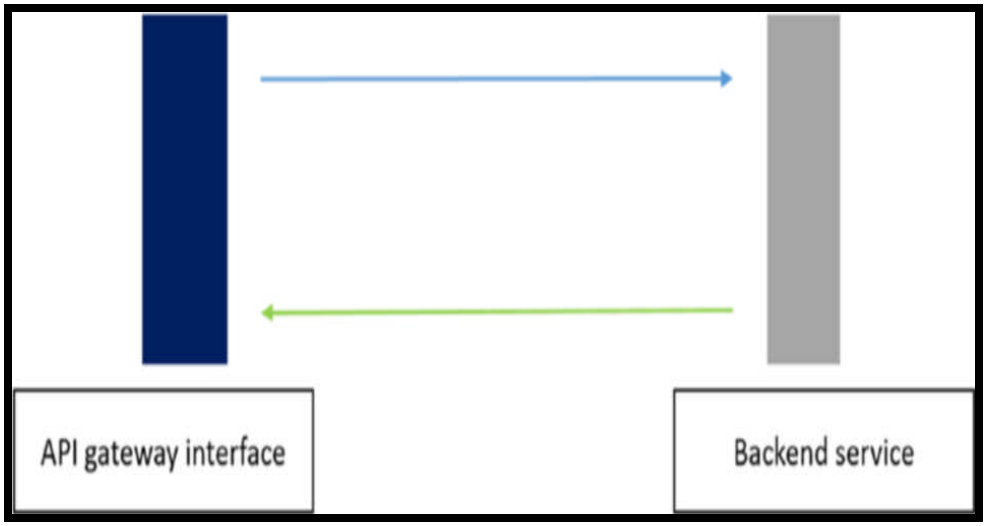
As discussed in the section 15.1 an API gateway is the single point of contact between a consumer and the backend services and requests. In this model, API gateways can have different mappings between backend service endpoints and API gateway interface endpoints.

There are four possible mappings and patterns related to these endpoints.

1. **One-to-One Mapping**

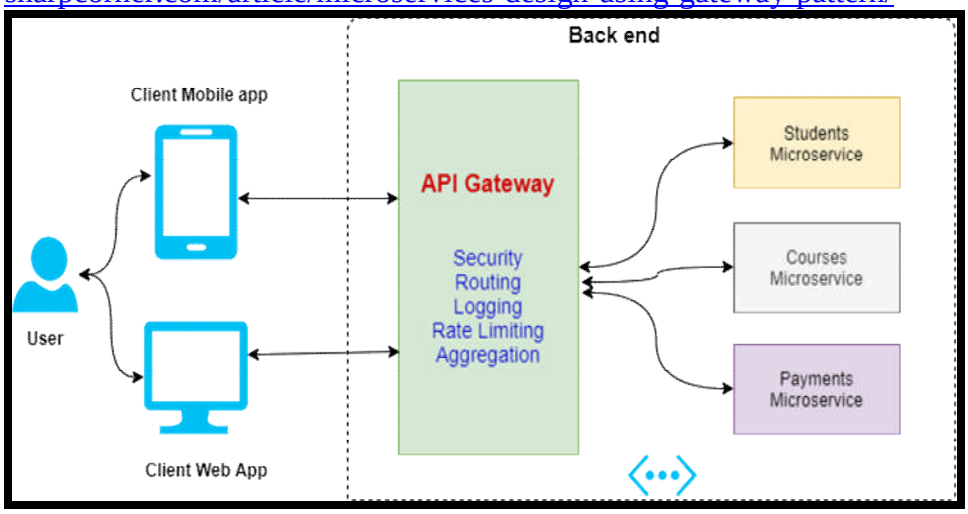
   In One-to-one mappings one backend service is mapped to one API gateway interface endpoint. Figure 15.2shows a one-to-one Mapping.

Figure15.2. One-to-One Mapping[1]



2. **One-to-Many Mapping**
   In this type of mapping one API gateway interface endpoint is mapped to many different endpoints of the various backend services. In this model a client can submits its request and the gateway can break it down into multiple requests, route them to different backend services, and return the response to the client in a single round-trip. An API gateway therefore reduces the number of requests between the client and server, improving latency and the user experience. In figure 15.3 a single request call can service micro services.

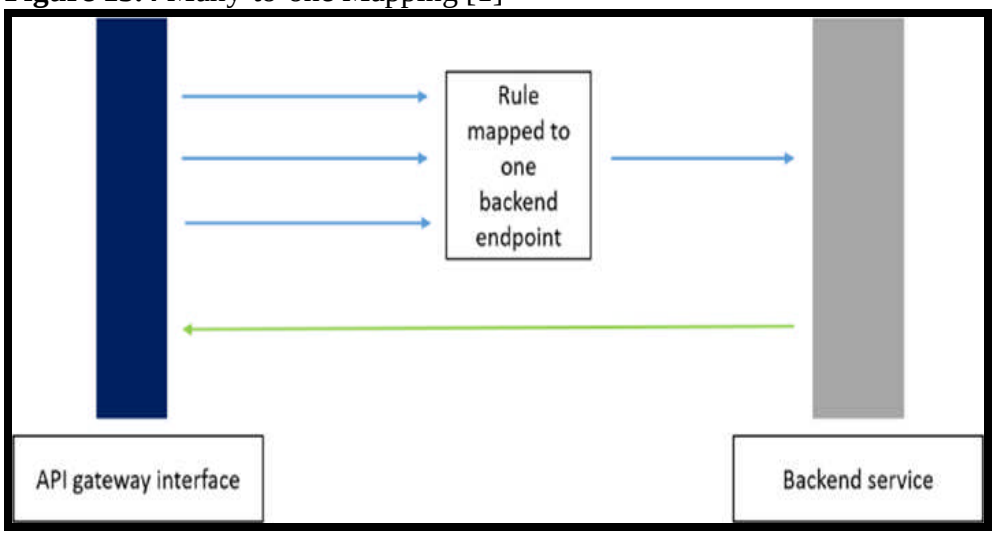
Figure 15.3 One-to-Many Mapping source https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/



3. **Many-to-One Mapping**
   This mapping is used as a business strategy. This model involves connecting multiple API gateway interface endpoints to one backend service endpoint. For example, an entity like a Product can have different states. Each state change of this Product entity will be an update operation. According to business logic each state change of this Product en-

tity can be a different operation in terms of the authorization and semantic meaning. So each API gateway interface has different endpoints with a meaningful URL, but they are mapped to a one-update endpoint of the backend service. Figure 15.4 shows this model.

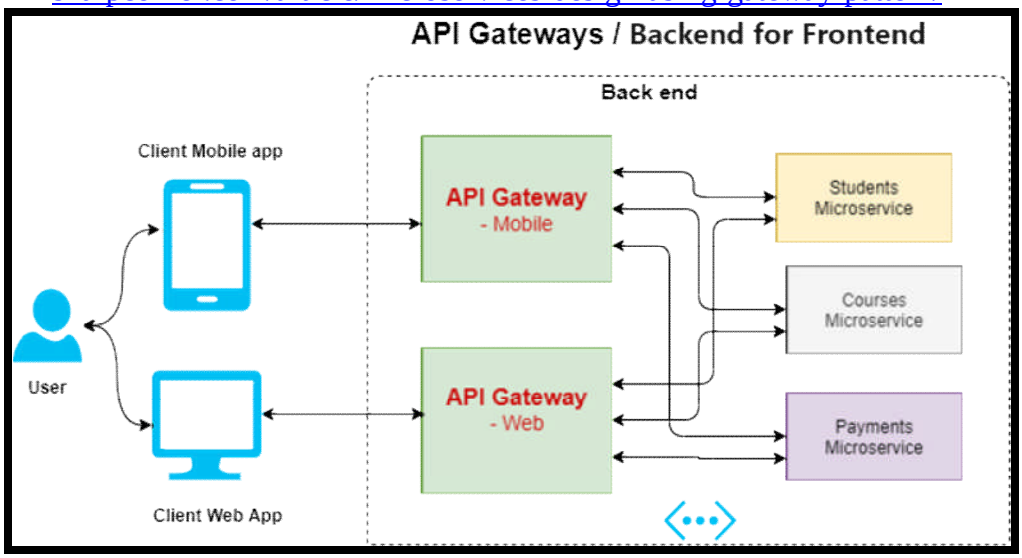
Figure 15.4 Many-to-one Mapping [1]



4. **Many-to-many mapping**

In this type of mapping, we can have many types of APIs mapped to multiple backend services. This however is a business decision where an organization might be offering a both a web application as well a Mobile based application to its clients as shown in Figure 15.5. Splitting into multiple gateways based on the client app type is referred to as "Backend for Front end" (BFF) pattern.

Figure15.5source:https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/

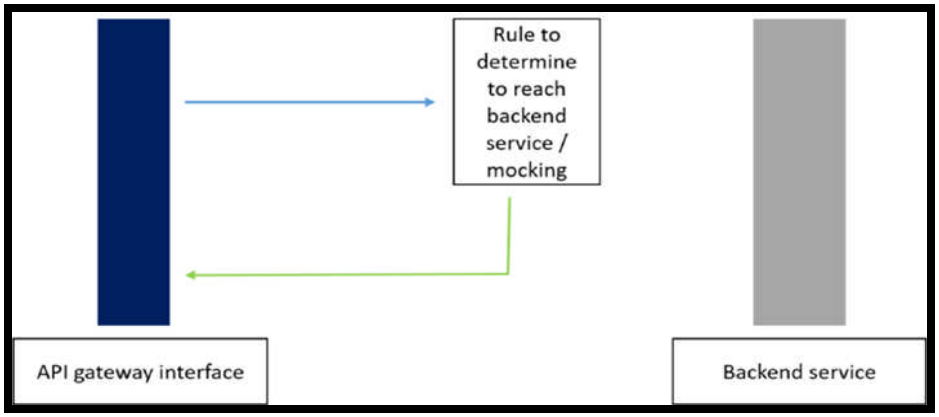


5. **One-to-None Mapping**

In this model shown in figure 15.6, a single API gateway interface is will be capable of producing a response without contacting that backend service. The API gateways here will be able handle requests, execute some fixed logic and return a response without connecting to a backend service. This kind of mapping is often done in mocking during development stage. A mock API server is useful during development and testing when live data is either unavailable or unreliable.

Figure 15.6[1]

.



15.3 AZURE API MANAGEMENT

Microsoft Azure API Management (APIM) is a way to create consistent and modern API gateways for existing back-end services. It helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services. Businesses everywhere are looking to extend their operations as a digital platform, creating new channels, finding new customers and driving deeper engagement with existing ones.

API Management provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security, and protection. One can take any backend and add the Azure API Management to launch a full-fledged API program based on it.

The Azure API Management system is made up of the following components:

- The **API gateway** is the endpoint that:
  - Accepts API calls and routes them to the back ends.
  - Verifies API keys, JWT tokens, certificates, and other credentials.
  - Enforces usage quotas and rate limits.
  - Transforms API on the fly without code modifications.
  - Caches backend responses where set up.
  - Logs call metadata for analytics purposes.
- The **Azure publisher portal** is the administrative interface where one-can set up their API program. The portal can be used to:
  - Define or import API schema.

- Package APIs into products.
- Set up policies like quotas or transformations on the APIs.
- Get insights from analytics.
- Manage users.

- The **Developer portal** serves as the main web presence for developers, where they can:
  - Read API documentation.
  - Try out an API via the interactive console.
  - Create an account and subscribe to get API keys.
  - Access analytics on their own usage.

When creating an Azure API Management service instance, Azure provides both a API gateway and a portal. API Gateway is the core engine, receiving requests, processing them, connecting to the backend service, and responding to requests.

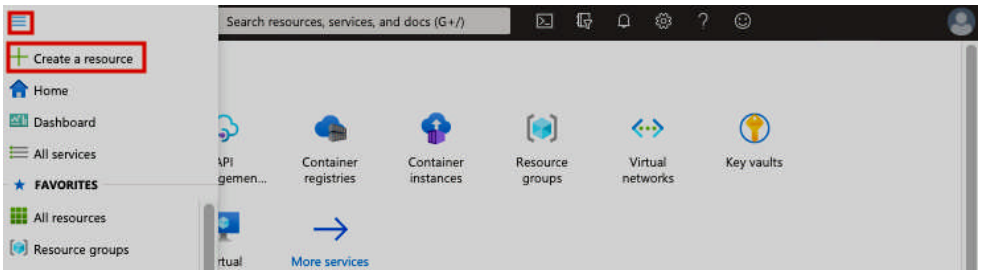
The publisher portal provides an administration interface to configure the API gateway and developer portal. The developer portal includes the interface and workflows for developer on boarding, API subscriptions, and other developer experience–related features.

15.3.1Creating an Azure API Management Service

To create an Azure API Management Service, navigate to the Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

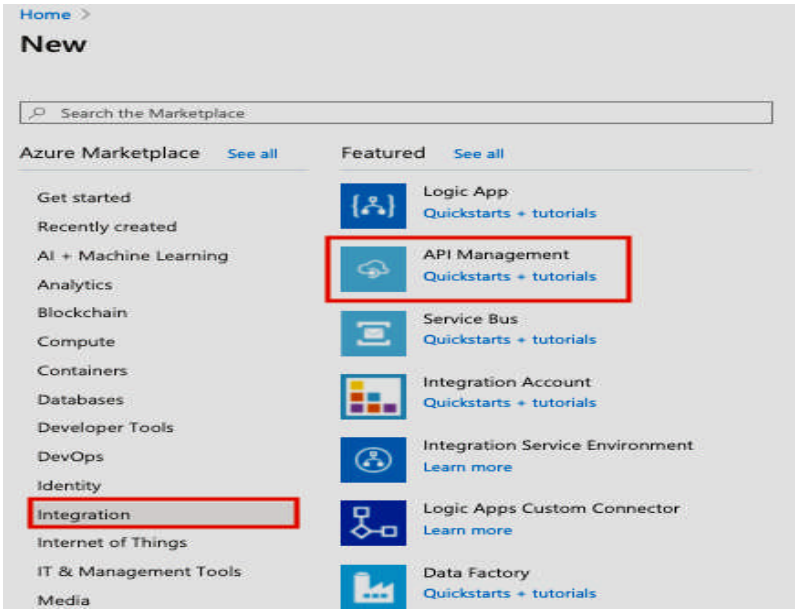
1.Sign in to the Azure portal.
2.From the Azure portal menu, select **Create a resource**.

Figure 15. 7 Create resource source:https://docs.microsoft.com/en-us/azure/api-management/get-started-create-service-instance



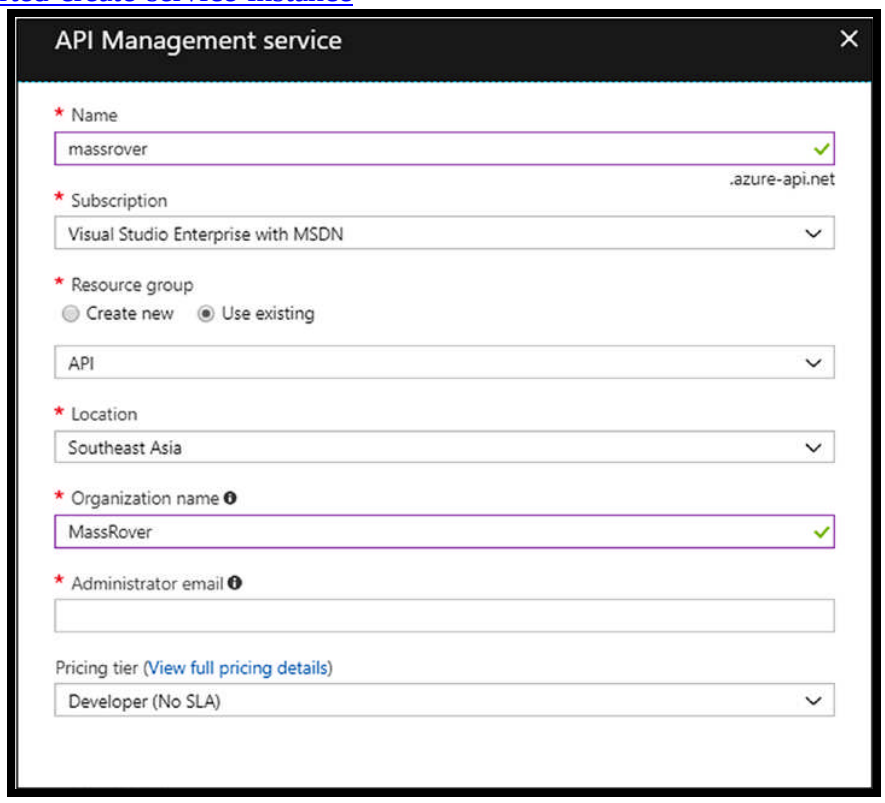
3.On the **New** page, select **Integration** > **API Management**.
Now look for “API Management,”and select API Management to create a service instance.

Figure 15. 8 New API Management Page
source:https://docs.microsoft.com/en-us/azure/api-management/get-started-create-service-instance



4. In the **API Management service** page, enter settings as shown in figure 15.9

Figure15.9 New Azure API Management service instance creation Source:https://docs.microsoft.com/en-us/azure/api-management/get-started-create-service-instance



In the Azure API Management creation window give a name. This name sets the URL of the API gateway and portal, which will appear as "newname.portal.azure-api.net". Here new name is the name that has been

set in the above dialog box. The DNS can be configured for this URL at a later stage. The URL “newname.portal.azure-api.net/admin” will open the publisher portal.

Next select the subscription and resource group (or create a new resource group), and select the location. Then specify the organization name. This name will appear in the developer portal as the organization that publishes the API.

Now specify the email address of the administrator which will be the emailed of the user who creates the service instance. The next step in this process is to select the pricing tier. The Developer tier is the most comprehensive offering, with sufficient request/response limitations in dev/test scenarios. Once the process of filling the form is complete the Azure API Management service instance will be created.

15.4 AWS API GATEWAY

API Gateway is the commercial name of the API management service offered in AWS. Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. Using API Gateway one can create a RESTful API or a Web Socket API, which will enable real-time two-way communication applications. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls. It also performs traffic management. It manages security through authorization and access control, throttling. It also performs monitoring, and API version management.
The following are some of the features of AWS API Gateway:

- Caching
- Request control
- Authentications
- Mocking(connecting to the backend service)
- API publishing via AWS Marketplace.

AWS API Gateway is tightly coupled with other AWS services like VPC and Lambda. It also allows client SDK generation for popular platforms like IOS and Android.

15.4.1Creating an AWS API Gateway Service

To create an AWS API Gateway Service, one should have an AWS account with IAM permission to perform actions. Otherwise, setup an account by visiting the page aws.amazon.com and choose “Create an AWS Account”. Amazon recommends that one should create an AWS Identity and Access Management (IAM) user with administrator permissions as a best practice. For more information on how to create an IAM user refer tohttps://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started_create-admin-group.html

1. Next log in to the AWS API Gateway console at https://console.aws.amazon.com/apigateway, and search for API Gateway option as shown in15.10 below; click on it to create an AWS Gateway instance as shown in figure 15.11 below.

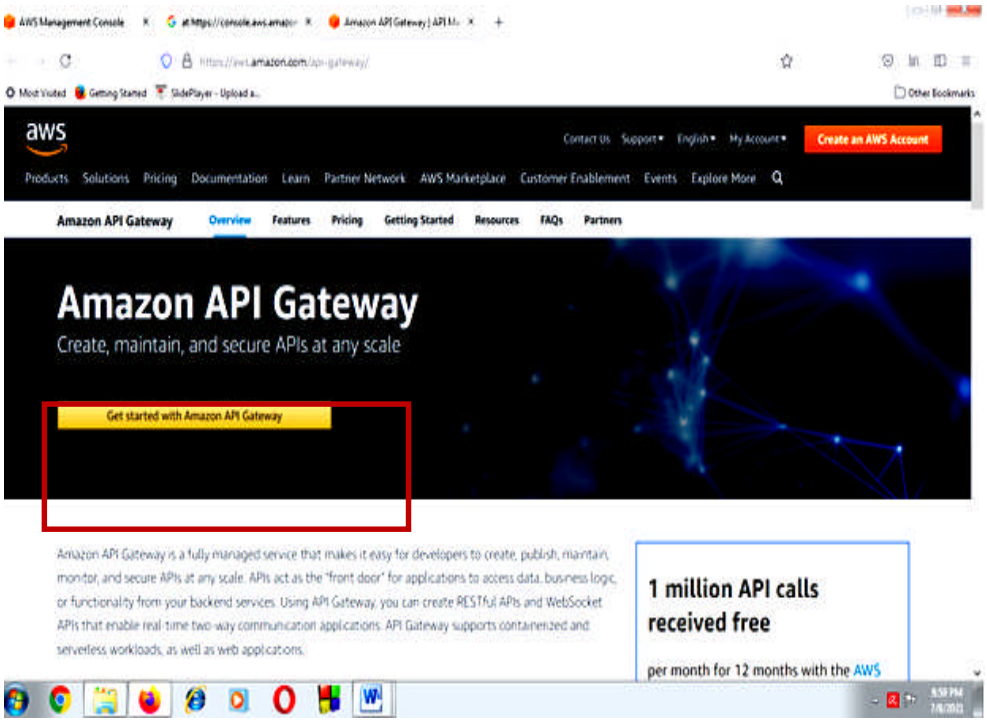
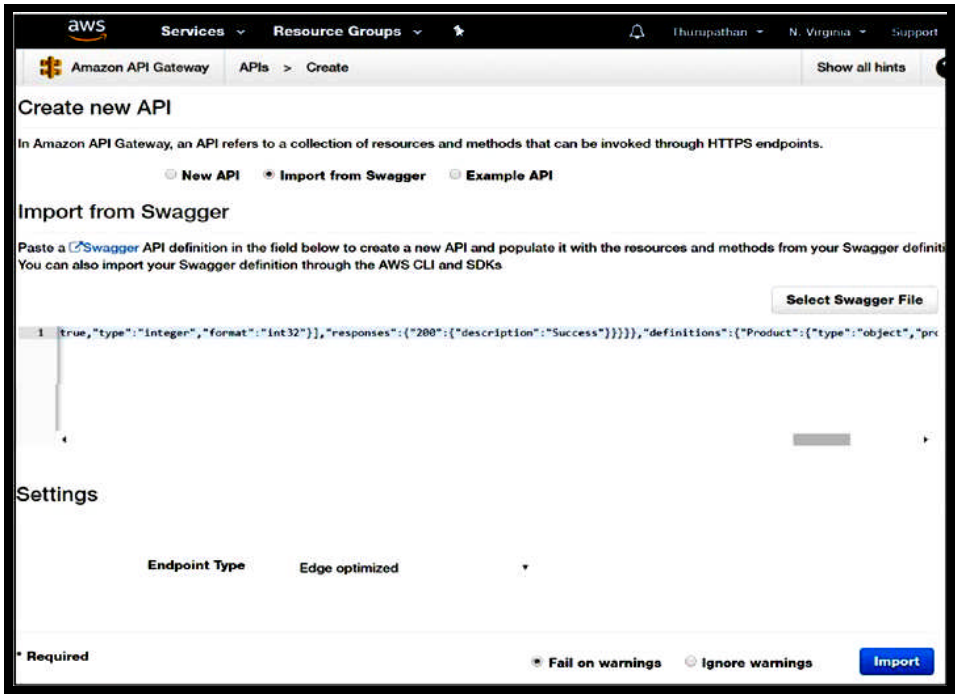


Figure 15.11 Create a new API in Amazon API gateway source:[1]



2.In the above window, there are three options for creating an API. If there is an existing API click on “Import from Swagger” and copy the Swagger

definition of the target API in the panel as shown in figure15.10, or upload from the definition file and click on Import to finish importing the file. One can also use an example API implementation option in AWS.

3. In the Settings section, select the endpoint type. There are two types of endpoints.

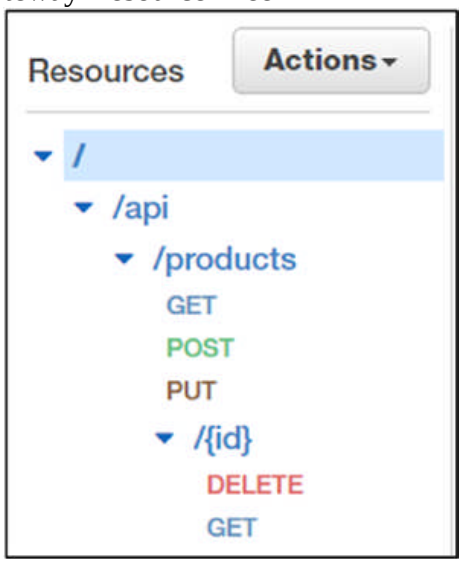
- Edge optimized: This is the default option, which enables the AWS Cloud Front distribution and improves connection time. This is a good choice in most cases. API requests from clients will be routed to the nearest Cloud Front edge servers across AWS regions just like in CDN.

- Regional: This option will route client requests to the region-specific API gateway, bypassing CloudFront distribution. In normal requests a request from the same region has the benefit of avoiding a round trip to Cloud Front thereby increasing performance. But if the requests originate from other regions, then they may experience delay. Deploying region-specific API gateways in target regions can solve this problem.

Once the API definition file is imported, the API will be visible in the panel. AWS API Gateway structures APIs as resources and methods. The API URI segment is a resource, and HTTP actions are referred to as methods.

In a URI segment like api/products, API and products are two different resources. A resource can have other resources and methods, and path parameters can also be added as resources. Curly braces are used to indicate the path parameters as shown in figure 15.11.API resources are organized in a resource tree according to the application logic as shown in figure 15.11. Each API resource can expose one or more API methods that have unique HTTP verbs supported by API Gateway.

Figure 15.12API gateway Resource Tree



One can select a resource and add a resource or method to it using the Actions drop-down menu on top. One needs to use curly braces when adding a path parameter as a resource. Click on methods like GET, POST, DELETE or PUT to configure it.

3. In the **Resources** pane as shown in figure 15.12, choose the resource root, represented by a single forward slash (/), and then choose **Create Resource**. For **Resource Name**, enter the name of the target API (Let's call it Sample API Proxy) being used.

4. In the next step create a GET method that enables the AWS service proxy to interact with the AWS service.

- **To create the GET method**
  1. In the **Resources** pane, choose /Sample API Proxy, and then choose **Create Method**.
  2. For the HTTP method, choose **GET**, and then save the choice.

5. In the next step we need to create the AWS service proxy execution role. For this one has to create an IAM role that the AWS service proxy uses to interact with the AWS service. We need to create an IAM role as an AWS service proxy execution role. Without this role, API Gateway cannot interact with the AWS service. To finish this step, sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/ . Choose Policies, go to Welcome to Managed Policies page, choose "Get Started". A list of policies will appear. Choose the policies and then choose "Create Policy".

5.Next choose **JSON** and then enter the following text.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Resource": [
    "*"
   ],
   "Action": [
    "sns:ListTopics"
   ]
  }
 ]
}
```

6.Now choose **Review policy**.

- Enter a name and description for the policy.
- Choose **Create policy**.

7. Next choose **Roles**- Choose **Create Role**.
8. Choose **AWS Service** under **Select type of trusted entity** and then choose **API Gateway**.
9. Choose **Next: Permissions**.
10. Choose **Next: Tags**.
11. Choose **Next: Review**.
12. For **Role Name**, enter a name for the execution role (for example, API Gateway AWS Proxy Exec Role), optionally enter a description for this role, and then choose **Create role**.
13. In the **Roles** list, choose the role you just created. You may need to scroll down the list.
14. For the selected role, choose **Attach policies**.
15. Select the check box next to the policy you created earlier (for example, API Gateway AWS Proxy Exec Policy) and choose **Attach policy**.
16. The role you just created has the following trust relationship that enables API Gateway assume to role for any actions permitted by the attached policies:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
    "Service": "apigateway.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

17. For **Role ARN**, note of the Amazon Resource Name (ARN) for the execution role. This will be required later. The ARN should look similar to:

```
arn:aws:iam::123456789022:role/APIGatewayAWSProxyExecRole
```

where 123456789022 is the AWS account ID.

18. In this step, specify the settings for the GET method so that it can interact with an AWS service through an AWS service proxy. Then test the method.

To specify settings for the GET method and then test it

1. In the API Gateway console, in the **Resources** pane for the API named Sample API Proxy, in **/** Sample API Proxy, choose **GET**.
2. Choose **Integration Request**, and then choose **AWS Service**.

3. For **AWS Region**, choose the name of the AWS Region where you want to get the Amazon SNS topics.
4. For **AWS Service**, choose **SNS**.
5. For **HTTP method**, choose **GET**.
6. For **Action**, enter List Topics.
7. For **Execution Role**, enter the ARN for the execution role.
8. Leave **Path Override** blank.
9. Choose **Save**.
10. In the **Method Execution** pane, in the **Client** box, choose **TEST**, and then choose **Test**. If successful, **Response Body** displays a response similar to the following:

```
{
  "ListTopicsResponse": {
    "ListTopicsResult": {
      "NextToken": null,
      "Topics": [
        {
          "TopicArn":                    "arn:aws:sns:us-east-
1:80398EXAMPLE:MySNSTopic-1"
        },
        {
          "TopicArn":                    "arn:aws:sns:us-east-
1:80398EXAMPLE:MySNSTopic-2"
        },
        ...
        {
          "TopicArn":                    "arn:aws:sns:us-east-
1:80398EXAMPLE:MySNSTopic-N"
        }
      ]
    },
    "ResponseMetadata": {
      "RequestId": "abc1de23-45fa-6789-b0c1-d2e345fa6b78"
    }
  }
}
```

19.Step 5: In the final step, deploy the API so that you can call it from outside of the API Gateway console. To do that, choose Deploy API in the Resources pane.

1. For Deployment stage, choose test.
2. For Deployment description, enter Calling AWS service proxy walkthrough.
3. Choose Deploy.

20. Test the API-To test go outside of the API Gateway console and use AWS service proxy to interact with the Amazon SNS service.

1. In the **Stage Editor** pane, next to **Invoke URL**, copy the URL to the clipboard. The following is the URL:

```
https://my-api-id.execute-api.region-id.amazonaws.com/test
```

2. Paste the above URL into the address box of a new browser tab as shown below:

```
https://my-api-id.execute-api.region-
id.amazonaws.com/test/mydemoawsproxy
```

3. When the URL is opened in the Browser, the following information will be displayed:

```
{"ListTopicsResponse":{
```

15.5 API SECURITY

Security is a major consideration in Public APIs as they expose the data and business operations to external consumers. As an API developer one should protect APIs from unauthorized consumers as well as control the consumption rate. They should incorporate mechanisms to protect the data.

A consumer of the API will need a trusted environment with policies for authentication and authorization. Some of the most common ways to strengthen API security are:

- **Using of tokens**. Tokens are a great way to strength security. This process involves establishing trusted identities and then controlling access to services and resources by using tokens assigned to those identities.
- **Using encryption and digital signatures**. Another way to secure API is to encrypt data using a method like TLS. Impose digital signatures to ensure that the right users are decrypting and modifying data, and no one else.
- **Identifying vulnerabilities**. Check the operating system, network, drivers, and API components for any vulnerability. Any weak spots that could be used to break into the APIs should be identified and fixed. Lastly sniffers should be used to detect security issues and track data leaks.
- **Using quotas and throttling**. To provide security quotas on how often API can be called should be imposed and its usage should be constantly monitored. More calls on an API may indicate that it is being abused. It could also be a programming mistake such as calling the API in an endless loop. Make rules for throttling to protect your APIs from spikes and Denial-of-Service attacks.

- **Use an API gateway**. API gateways act as the major point of enforcement for API traffic. A good gateway will allow you to authenticate traffic as well as control and analyze how your APIs are used.

Implementing API security in public cloud is a very challenging task and involves providing security at multiple levels of a system. There two important ways of implementing API security:

- Request-based security implementations –This security implementation dictates the policies and constraints on API consumption. It lays down the rules for API consumption like who can consume the API and how much.
- Authentication- This states the policies and constraints on authentication and authorization like who the consumer is and what the consumer can access.

This chapter focuses on how the above two aspects of API security and how they are implemented in Microsoft Azure and Amazon's AWS.

15.6 REQUEST-BASED SECURITY

Request-based security implementation is used to identify a consumer and then apply some constraints on their consumption. This can be achieved either by limiting the consumption rate or allowing or blocking the consumer. Limiting the API consumption rate is important because more calls on an API may indicate that it is being abused. It also protects the APIs from a spike in usage and prevent Denial-of-Service attacks. Sometimes excessive calls to an API may also be an indicator of a programming mistake such as calling the API in an endless loop.

In Request-based security implementation, a consumer is identified through a very simple mechanism like API keys and his consumption rate is tracked. In the direct selling model of API economy, the consumption rate of an API is a basic parameter in defining different API SKUs. Good Cloud based API management services such as Azure API Management and AWS API Gateway have built-in settings to configure and implement request-based security rules.

15.6.1 Request-Based Securityin Azureapi Management

Azure API Management manages API security by configuring the request-based security settings. The rules for configuring the request-based security settings are applied using Azure API Management policies.

Request-based security in Azure API Management can be implemented through:

- **Subscriptions and Subscription Keys**
- **Request Rate Limits**

- Quota Limits
- IP restrictions

15.6.1.1Subscriptions and Subscription Keys

In Azure API Management, a consumer having a valid subscription key associated with an Azure API Management product can only make a successful request to that API product. API developers can request or automatically retrieve (depending on the setting) subscription keys for products in the developer portal.

A single subscription to a product provides two subscription keys: A primary key and a secondary key. Having two subscription keys helps in decreasing the downtime during key rollover. The secondary key can be used to replace the primary key that has been compromised or is vulnerable to compromise.

Usage is tracked at the subscription level regardless of which key is used. Generally, subscription keys are used to track a consumer's usage. But Azure API Management allows developers to use other ways to configure usage limits based on parameters like IP address or response codes etc.

Subscription keys are sent to the API Azure API Management gateway in the request header as **ocp-apim-subscriptionkey.**

15.6.1.2 Request Rate Limits

Request rate limit is another mechanism through which we can implement security in Azure API Management.
To set the Request Rate Limits follow the steps below:

1. Navigate to the Azure API Management service and select the Azure API Management product.
2. Then navigate to the policies section of the product and add the rate-limiting policies.

Product-level policies are applied to all the endpoints in an aggregated manner. If the rate is 10 requests per minute for a product and the product has three such API endpoints, the consumer can make a maximum of 10 requests per minute as a combined request rate. The consumer is not allowed to make 10requests to each endpoint separately. Generally request-based policies are applied at the product level.

Endpoint-level policies can be applied to enforce such rules for rate-limiting.

For example, for any given product the following policy statement in the inbound section allows 4000calls in 90 seconds for the API endpoints included in the specific product.

```
<rate-limit calls="4000" renewal-period="90" />
```

If the caller exceeds this limit, Azure API Management will respond with a 429 HTTP code which indicates “too many requests”. The response body will contain the message and the retry time period. The caller must wait for that time period before sending the next request. Specifying the remaining time in response body will help in resending the requests to gateway at specified retry time period rather than sending requests at random intervals.

We can also control the request rate using an arbitrary key value. By setting the key value to any variable like IP Address which is a variable value in the request message sent to and which is accessed by Azure API Management gateway, we can control or restrict the request rate. In the example below, we can set the request rate limit by setting the IP address in the counter-key which is some arbitrary key value.

```
<rate-limit-by-key calls="1200"renewal-period="90"
increment-condition="@(context.Response.StatusCode == 200)"
counter-key="@(context.Request.IpAddress)"/>
```

The above policy allows a caller with a particular IP address to make1200 requests. If the calls exceed this limit the user will have to wait for 90 seconds before sending the next request.

15.6.1.3 Quota Limits

Having Quota Limits is another way to ensure security in Azure API Management. Quota limits are enforced to control request quotas. Applying Quota limits is a way to monetization. Here different SKUs can have different quotas limits which can be applied through quota policies. For example, in the below example we can have a quota policy based on the number calls or the bandwidth. When the number of calls or the bandwidth exceeds the specified limit (whichever limit exceeds first), there will be a waiting period of
4800 before resending a request call.

```
<quota calls="120000" bandwidth="100000" renewal-period="4800"
/>
```

This policy tracks API usage based on the subscription key. Quota limits are also based on arbitrary keys just like request rate limits.

15.6.1.4IP restrictions

This final section on security in Azure API Management is based IP restrictions. Here IP-based restriction policies will be used to restrict certain IP addresses.

```
<ip-filter action="allow">
<address-range from="10.1.1.4" to="10.1.1.14" />
</ip-filter>
```

The above policy will admit IP addresses between 10.1.1.4 and 10.1.1.14.

15.6.2Request-based security in AWSAPIGateway

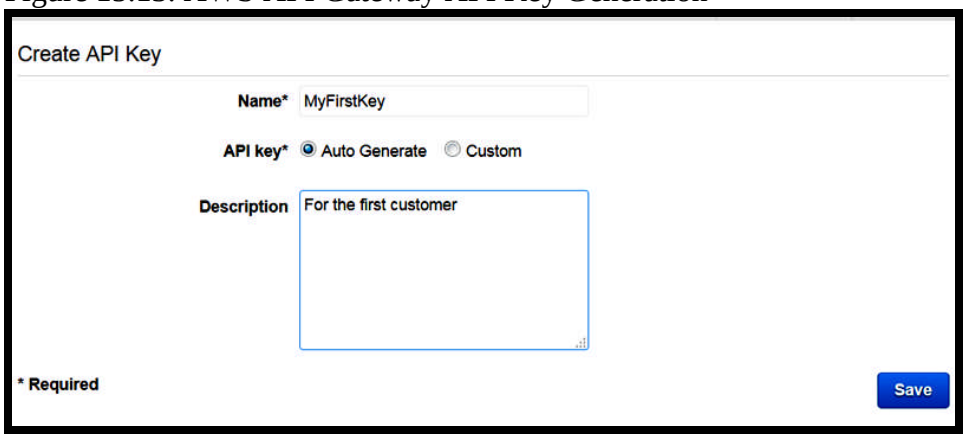
Request-basedsecurity in AWS API Gateway is implemented through the following:

- API Keys
- Rate Limits
- Quota Limits

15.6.2.1 API Keys

Keys for the AWS APIs canbe auto generated by the AWS API Gateway service or custom made as shown in figure 15.13. API usage plans for a customer are tracked with the help of these API keys.

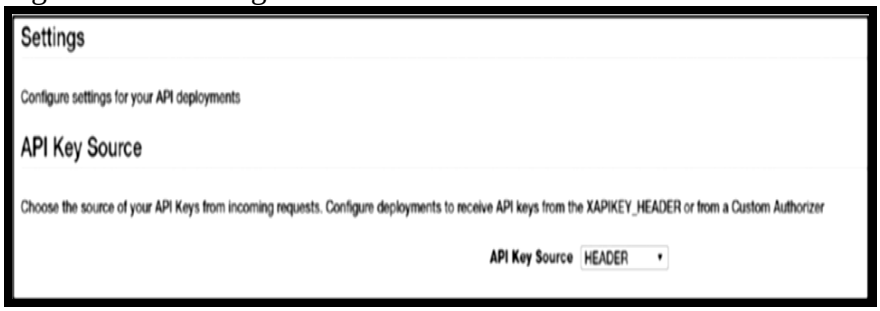
Figure 15.13. AWS API Gateway API Key Generation



One can customize how consumers send API keys to the AWS API Gateway. API keys are sent in the HTTP request header X**APIKEY_HEADER** as shown below**.** They can also be sent through configured custom authorizers. One can choose the API Key Source as Header or as custom authorizers from the drop-down shown in figure 15.14.

```
API Keys in HTTP request header
{“api-key”:”9038-909-0998-9999”}
```

Figure 15.14 Settings



The configuration of the API can be done under the Settings section of the selected API as shown in figure 15.14

15.6.2.2 Rate Limits

The Requests to an API can be limited by setting the rate limit. We can set the usage plans for a customer with the help of API Key associated with an API user. These keys identify each user of the API, and allow the API developer to control the set of services and service stages (environments such as test, beta, and production) that the key holder can access. The rate limits can be set in the usage plans, and requests can be tracked via the API keys associated with that customer. One usage plan can include many APIs, and all the APIs in a usage plan are constrained by the rate limit.

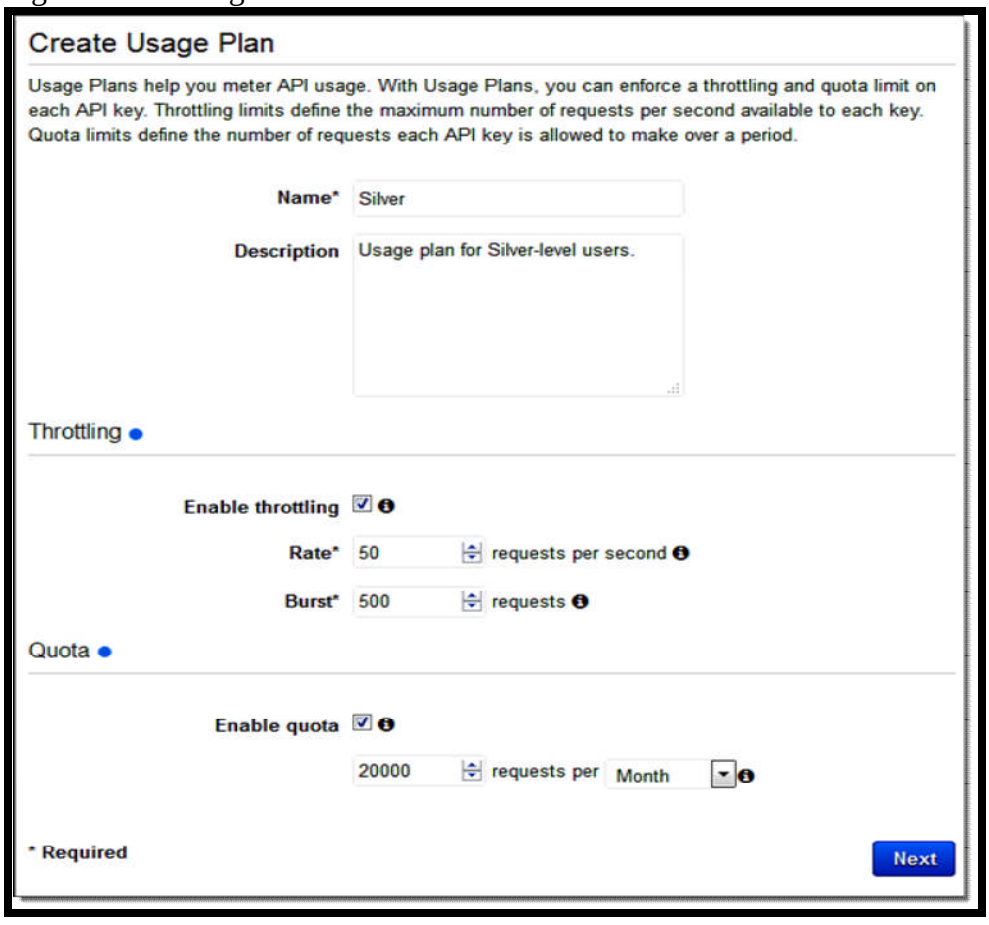
In a usage plans the rate limit can be set via the following two parameters:

- T**hrottling**–Overall request rate (average requests per second) and a burst capacity.
- **Quota**- Number of requests that can be made per day, week, or month.

To set the rate limits by creating a usage plan, follow the following steps:

- Navigate to the usage plan.
- Set values for Throttling and Quota as shown in Figure 15.14.

Figure 15-14 Usage Plan



.

In the above usage plan, throttling has been enabled. The rate is set to 50requests per second which means that this particular usage plan can handle on an average 50 requests per second without any throttling. The burst is 500 requests the usage plan can accommodate when the requests arrive simultaneously**.**

The quota limit is set along with throttling while creating a usage plan. In figure 15.14 above, the quota limit is20000 which is the number of requests per month.

15.7 AUTHENTICATION AND AUTHORIZATION

All APIs implement authentication and authorization. Authentication identifies a caller of an API and authorization provides information on whether the caller has the credentials to access to the secured resources or not.

- **Authentication** is the verification of the credentials of the connection attempt. This process consists of sending the credentials from the remote access client to the remote access server in an either plaintext or encrypted form by using an authentication protocol [3].
- **Authorization** is the verification that the connection attempt is allowed. Authorization occurs after successful authentication [3].

An API can implement authentication and authorization in many different ways. One of the most basic ways to do this is to send the required security information as value in the request header to an API gateway. A special HTTP header containing the username and password can be added in the request header. But it is very easy to retrieve this information basic authentication. in a HTTP protocol; but can be sent securely while using a SSL or TLS protocol.

There are many standards and protocols for securing an API. Some of these standards and protocols are pretty common and are widely used. Others are highly vendor specific.

This section will discuss the following most common cloud-based API authentication and authorization standards. Some of these standards are specific to Microsoft Azure and Amazon's AWS.

- API Keys
- OpenID and OAuth
- Securing APIs with Azure Active Directory V2
- Issuing Custom JWT Tokens
- Pre-Authentication in Azure API Management
- Authorizers in AWS API Gateway

15.7.1API Keys

As discussed in the preceding sections, API key is a unique identifier used to authenticate a user, developer, or calling program to an API. However, they are typically used to authenticate a project with the API rather than a human use. A developer involved in developing an API based

project will register with the API provider or provide identity to an API provider to obtain a key to access the API.

For example, developers using the Google Maps' API have to prove their identity via Google login. Post the verification of their Google account, they receive the API key. This API key is a random string of characters; each request should contain this key. The API key will identify the caller of the API and determine the access limits. Generally, this kind of authentication is used when we need to identify the API usage rather than the API user as in the direct-selling API model.

The full key management of Public APIs is controlled by the API or handled at the gateway service. In Azure API Management subscription keys and AWS API Gateway API keys, the issuing and management of the keys is handled at the gateway service, and the backend service is unaware of the key. In the direct-selling API model, most keys do not expire, but may be invalidated by the API due to too many requests, suspected request-based attacks, or missed payments. However, API consumers are not allowed to recycle their keys for obvious security reasons

To get the API keys from a API provider, he needs to follow the below steps:

1. First the developer requests an API key from the API/API provider. This request may have preconditions, like certain validations or payments.
2. Next when the validations or payments are verified, the developer receives the key from the API/API provider
3. The developer stores this key in the application.
4. Finally when sending a request to the API, the application sends the key to the API in the request string.

The same key can be used by the developer across different applications as long as the key is valid in terms of usage conditions and limitations. In Azure and AWS, the gateway service handles key management. This offloads the key management logic from the API implementation.

Open ID and OAuth standards

Open ID and OAuth are two security standards for authentication and authorization in APIs security implementation. OpenID is for authentication purpose and OAuth is for authorization.

When performing authorization, authentication is also done. This means that in OAuth implementation, authentication has to be performed through some other standard such as OpenID.

Both OpenID and OAuth are implemented via browser redirects for example several web sites give new users the facility to login through their preexisting Google or Facebook accounts, instead of creating a new account on their web site. Google or Facebook will authenticate the user to

this web site. This is possible because of OpenID standard and it facilitates the user authentication from the same provider to multiple entities.

Sometimes Facebook may respond with a prompt, saying that some website is trying to access a user's email, photos, or some other information. Here the website which is trying to access user's data on Facebook, requests the user's permission by prompting to allow access to data stored in Facebook. By agreeing to this request the user is allowing website to read his/her data. Facebook gives some access to this website to make requests to its resources and if the permission is granted, future communication can take place between to the website and Facebook based on the allowed permissions. All this is OAuth implementation.

15.7.3 Securing APIs with Azure Active Directory V2

Azure Active Directory (AAD) is the common way to secure services in Azure application development. Azure Active Directory (Azure AD) is Microsoft's cloud-based identity and access management service, which helps employees sign in and access resources in:

- External resources, such as Microsoft 365, the Azure portal, and thousands of other SaaS applications.
- Internal resources, such as apps on corporate network and intranet, along with any cloud applications developed by an organization

This section focuses on how AAD is used by developers as a standards-based approach for adding single sign-on (SSO) to their application, allowing it to work with a user's pre-existing credentials. The new version of AAD known as AAD V2, is the revised version of AAD V1with some significant changes.

AAD V2 is used for authenticating users of an API service. To authenticate the identity of a user of an API an identity provider is used.

Whenever there a user wants to use the services of an API, the following steps are used:

1. The client first obtains a valid identity from the identity provider.
2. Next, the obtained identity is sent to the API in the request
3. API validates the identity and serves the request.

In order to validate the identity, the API should know the identity provider and the mechanism used to validate the identity.

There are two major steps involved in authenticating the user of an API:
1. Mechanism for the client to obtain the token (the identity)
2. Mechanism for API to validate the token

First, create the setup for the client to request a token from the AAD v2. The mechanism for the client to obtain the token is done via the

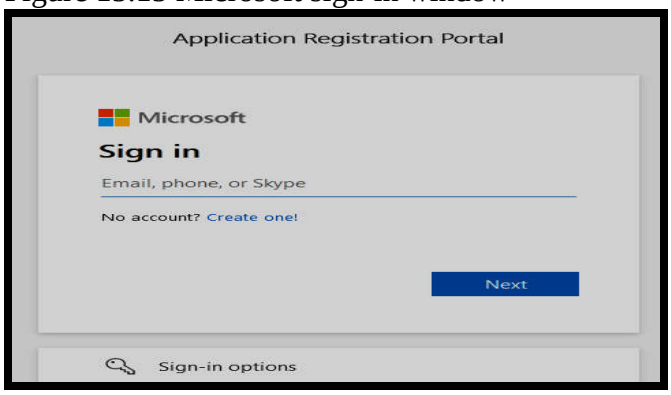
AAD v2 OAuth authorize endpoint. The client must make a request to this authorize endpoint, submit their credentials, and obtain the OpenID. For example, in the URL below a request is sent with a registered client ID and a configured redirect URI. The redirect URI is where the AAD will redirect the requested response type.

```
https://login.microsoftonline.com/common/oauth2/v2.0/
authorize?client_id=[client id]&response_type=id_
token&redirect_uri=[redirect uri]&scope=user.readopenid
profile&nonce=3c9d2ab9-2d3b-4
```

In order to register a client application in AAD and set up the information, perform the following steps:

1. Go to https://apps.dev.microsoft.com/ (which is the endpoint to register AAD v2 applications)
2. Sign in with a Microsoft account as shown in figure 15.15.

Figure 15.15 Microsoft sign-in window



3. Once signed –in navigate to the new Application Registration Portal tab in the menu bar.
4. Create a new AAD v2 application, by specifying- the client ID, the application platform as web and the redirect URL as shown in Figure 15.16 and 15.17.

Figure 15.16*AAD V2* new Application Registration

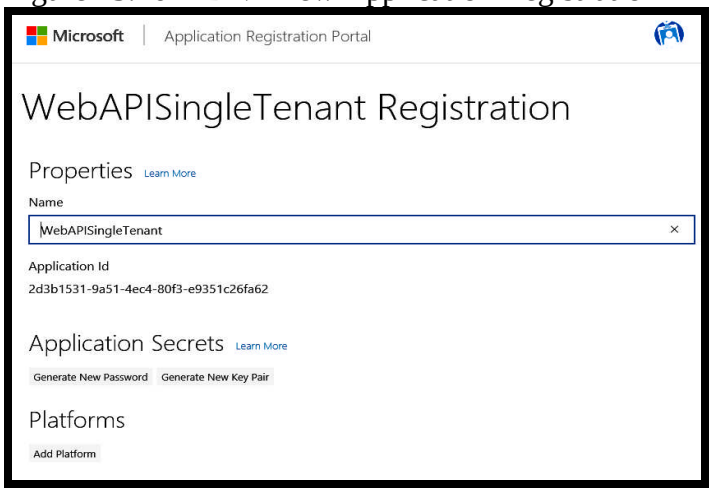
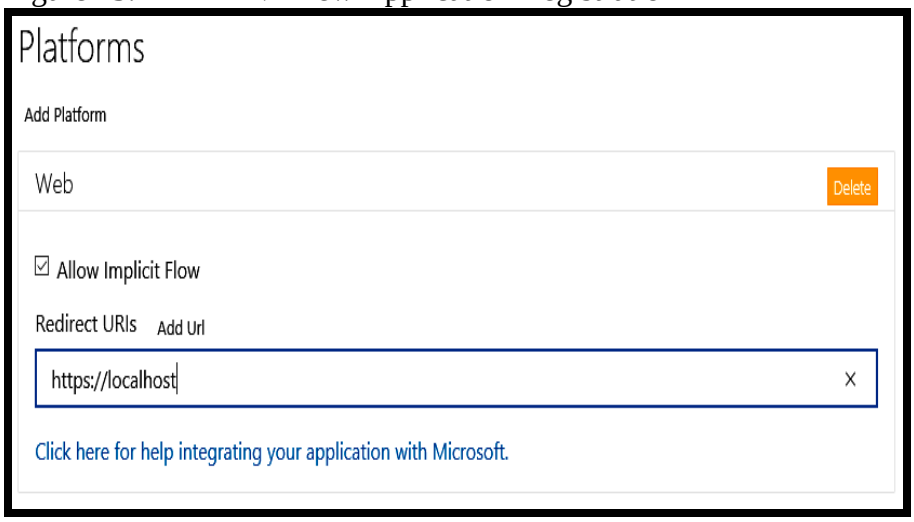


Figure 15.17 AAD V2 new Application Registration



The client should specify the client ID and the configured reply URL in the request, as shown below:

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=2d3b1531-9a51-4ec4-e9351c26fa62&response_type=id_token&redirect_uri=https:// local-host&scope=openid&nonce=3c9d2ab9-2d3b-4
```

The client ID and the redirect URI should match the values configuredin the AAD v2 application. When the above request is sent, the token is sent to the redirect URI https://localhost. The scope is open because the purpose of the request is to obtain the identity of the client.

The above request URL accepts any valid AAD authentication because the authorization URL points to the common endpoint. This applies to both Microsoft accounts and organizational accounts. If only the organizational authentication is acceptable, replace "common" with"organizations" as below:

```
https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize?client_id=2d3b1531-9a51-4ec4-e9351c26fa62&response_type=id_token&redirect_uri=https://localhost&scope=opened &nonce=3c9d2ab9-2d3b-4
```

If designing a single-tenant application, which expects authentication from one AAD tenant, the tenant ID can be specified in the request URL.

```
https://login.microsoftonline.com//[tenant id]//oauth2/v2.0/authorize?client_id=2d3b1531-9a51-4ec4-e9351c26fa62&response_type=id_token&redirect_uri=https://localhost&scope=opened &nonce=3c9d2ab9-2d3b-4
```

Clients can make any one of the above requests, authenticate themselves with AAD credentials, and obtain the OpenID information. The OpenID information will be in the id_token issued by the AAD v2 endpoint to the instructed redirect URL.

When URL below is tested for the first time a consent screen will appear.

This is the request from AAD v2 to get consent from the user and issue the OpenID information to the requesting client. Subsequent login attempts will not ask for this consent.

Request URL
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=2d3b1531-9a51-4ec4-e9351c26fa62&response_type=id_token&redirect_uri=https://localhost&scope=opened &nonce=3c9d2ab9-2d3b-4

After successfully logging in and obtaining consent, AAD v2 will redirect to the specified URL with the id_token, like below.

https://localhost/#id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUz
I1NiIsImtpZCI6IjFMVE16YWtpaGlSbGFfOHoyQkVKVlhlV01xbyJ9.eyJ2ZXI
iOiIyLjAiLCJpc3MiOiJodHRwczovL2xvZ2luLm1pY3Jvc29mdG9ubGluZS5jb
20vOTE4ODA0MGQtNmM2Ny00YzViLWIxMTItMzZhMzA0YjY2ZGFkL3YyLjAiL
CJz
dWIiOiJBQUFBQUFBQUFBQUFBQUFBQUFBQUFKNFk4NE56dWdlYl8yTFBWcFl
kbzN
jIiwiYXVkIjoiNzZkODg3NzktZDg4OC00MDFmLTg1NjUtMjMxYWVlMzg1YjE0Ii
wiZXhwIjoxNTIx-
NDY5NTE5LCJpYXQiOjE1MjEzODI4MTksIm5iZiI6MTUyMTM4M
jgxOSwidGlkI-
joiOTE4ODA0MGQtNmM2Ny00YzViLWIxMTItMzZhMzA0YjY2ZGFk
Iiwibm9uY2UiOiIzYzlkMmFiOS0yZDNiLTQiLCJhaW8iOiJEVEghayFwMzdNamh
jcE8qWkF6aTBKb3Z0b2x4RDhVZDk5R3Ria2RaSFAqbFRVU01wbUp1Q2h1IVVtbV
hzbEoqYkFkTWU3bEtTY2JYbjlIWTFNc0tUNUxBMEM5anJBdjBaTE1XV3oyZXVnS
mZSbGdRMGNZZWlnMHd5S0piakVldE13JCQifQ.HZbpQmkdi-2yOHtwtF-zFJhz7
RJe3_GIkcmS5u5EwIV7U_5x8S_2_o6JfQ0KBpnhzop5UijP99Rjan0dTtfat2Bs
TnWZloLbKy9X30XwJzrd-8WU2Nz7zwg24rMKEu0t6c8-
uR2ze-U1dhogGQZj6eu
rvnsedL4ET9eYehvPxV18U3AsSkZ2LAERpEiZeu16G0ORWpwGBI5NvogYhkRxzY
iZGlC5MsmkvZa4VdyVce_zJ-AnGrgwBvi5oL083RFGNMKUDevHDGpefO-UW3XOq
D3WiJmvdYx3g4ZPyamYH7UgyR0DIgOXnuLIoPXEop8AgrbJkPjJRbIkRVDMqjFAw
w

One can use any available JWT decoding tool to see the details. http://calebb.net/ [1] is a good online tool used to view the internals of JWT, and the above JWT is decoded as follows:

{
typ: "JWT",
alg: "RS256",
kid: "1LTMzakihiRla_8z2BEJVXeWMqo"

```
}.
{
ver: "2.0",
iss: "https://login.microsoftonline.com/9188040d-6c67-4c5b-b112-
36a304b66dad/v2.0",
sub: "AAAAAAAAAAAAAAAAAAAAAJ4Y84Nzugeb_2LPVpYdo3c",
aud: "76d88779-d888-401f-8565-231aee385b14",
exp: 1521469519,
iat: 1521382819,
nbf: 1521382819,
tid: "9188040d-6c67-4c5b-b112-36a304b66dad",
nonce: "3c9d2ab9-2d3b-4",
aio: "DTH!k!p37MjhcpO*ZAzi0JovtolxD8Ud99GtbkdZHP*lTUSMpm
JuChu!UmmXslJ*bAdMe7lKScbXn9HY1MsKT5LA0C9jrAv0ZLMWW
z2eugJfRlgQ0
cYeig0wyKJbjEetMw$$"
}.
```

After obtaining the token, clients will send it to the API, which should be able to validate the token and retrieve the information to be used in the business logic.

In an ASP.NET Core application, one can use the following code [1] to perform the token validation and query claims from the token.

```
private async Task<System.IdentityModel.Tokens.Jwt.
JwtSecurityToken>
ValidateAADIdTokenAsync(string idToken)
{
var stsDiscoveryEndpoint = "https://login.microsoftonline.
com/common/v2.0/.well-known/openid-configuration”;
var configRetriever = new Microsoft.IdentityModel.
Protocols.OpenIdConnect
.OpenIdConnectConfigurationRetriever();
var configManager = new Microsoft.IdentityModel.Protocols
.ConfigurationManager<OpenIdConnect
Configuration>
(stsDiscoveryEndpoint, configRetriever);
var config = await configManager.GetConfigurationAsync();
var tokenValidationParameters = new Microsoft.
IdentityModel.Tokens.TokenValidationParameters
{
IssuerSigningKeys = config.SigningKeys,
};
var tokenHandler = new JwtSecurityTokenHandler();
tokenHandler.ValidateToken(idToken,
tokenValidationParameters, out var validatedToken);
return validatedToken as JwtSecurityToken;
}
```

The method receives the token as a string and obtains the tokensigning keys from the provider (AAD v2) via the secure token serviceendpoint.

https://login.microsoftonline.com/common/v2.0/.wellknown/openid-configuration

This URL will be used by Open Id Configuration in order to obtain the signing information. You can change this URL by replacing “common” with “organizations” or “tenant id” as per the request URL pattern discussed earlier.

The obtained signing keys and other validation settings are used to create the Token Validation Parameters. Eventually the token is validated, and the output will be converted to a Jwt Security Token**,** which contains the claims in the token for programmatic access. The above code snippet **“**Token Validation Parameters” has a minimum validation configuration, meaning it checks whether the token is issued by the correct trusted entity by validating the signing keys. In the case of a real-world implementation, more complex validation rules would be used to perform the validation, along with signing rules such as validating the issuer, audience, and expiration.

In a practical implementation, the above token validation code snippet would be a filter in the ASP.NET Core. Each request header from the client would contain the id_token and be validated via the filter. Certain claims will be retrieved from the id_token in order to execute the business logic.

Public identity providers like Google and Facebook can be used to authenticate users. Most of the claims in the OpenID information cannot be related to the custom business logic of a typical enterprise line of business application, because the business logic deals mostly with application-specific roles and permissions, which are outside the context of the mentioned identity providers.

To solve this, we can issue the custom token from the application after validating the identity from the external providers (Google, Facebook etc.). In this mode, we depend on external identity providers for authentication and issue custom tokens for the authorization. The next section explains the process of issuing custom tokens using custom JWT Tokens.

15.7.4 Issuing Custom JWT Tokens

As explained in the previous section on Securing APIs with Azure Active Directory V2, there are certain cases where we need to create and issue custom JWT tokens. For example, say you're developing a SaaS application that relies on several identity providers. These identity providers help users authenticate to the application with less friction, and OpenID plays a key role in establishing a single sign-on experience, but once the user is authenticated, the application should be aware of the authorization information of the user, including roles and permissions.

In simple terms, Google or Facebook cannot store a user's details, even if the user is an admin of your custom application. It is the responsibility of the backend service/application to manage the authorization.

The previous section explains how to obtain a JWT token with OpenID claims from AAD and validate it. This would help secure the API using AAD from an authentication point of view, but once the user has logged in, API has to determine the authorization of the user in order to decide what the user can do inside the application i.e. the user specific roles and permissions in the application have to be authorized.

The below code snippet [1] shows how to issue a custom JWT token using a symmetric signing key.

```
private string IssueJwtToken(JwtSecurityTokenaadToken)
{
var msKey = GetTokenSignKey();
var msSigningCredentials = new Microsoft.IdentityModel.
Tokens.SigningCredentials
(msKey, SecurityAlgorithms.HmacSha256Signature);
var claimsIdentity = new ClaimsIdentity(new List<Claim>()
{
new Claim(ClaimTypes.NameIdentifier,
"thuru@massrover.com"),
new Claim(ClaimTypes.Role, "admin"),
}, "MassRover.Authentication");
var msSecurityTokenDescriptor = new Microsoft.
IdentityModel.Tokens
.SecurityTokenDescriptor()
{
Audience = "massrover.client",
Issuer = "massrover.authservice",
Subject = claimsIdentity,
Expires = DateTime.UtcNow.AddHours(8),
SigningCredentials = msSigningCredentials
};
var tokenHandler = new JwtSecurityTokenHandler();
var plainToken = tokenHandler.CreateToken(msSecurityToken
Descriptor);
var signedAndEncodedToken = tokenHandler.
WriteToken(plainToken);
return signedAndEncodedToken;
}
```

The above JWT issuing code obtains the signing key from this private-method[2].

```
private Microsoft.IdentityModel.Tokens.SymmetricSecurityKey
GetTokenSignKey()
{
var plainTextSecurityKey = "massrover secret key";
```

```
var msKey = new Microsoft.IdentityModel.Tokens.
SymmetricSecurityKey
(Encoding.UTF32.GetBytes(plainText
SecurityKey));
return msKey;
}
```

The token-issuing code uses Claims Identity to include application-specific claims in the token subject. For the next step, “Microsoft. Identity Model. Tokens. Security Token Descriptor” is used to construct a full JWT token along with the custom claims in the subject.

This object is used to create the JWT token using the Create Token method from JWT Security Token Handler, and the token is returned as a string. When this custom token is issued, the API should be able to validate the token as well, when it is returned from the callers in the requests.

The below code snippet [1] shows the token validation code.

```
public bool Validate Mass Rover Token (string token)
{
var token Validation Parameters = new Microsoft.IdentityModel.Tokens
.TokenValidationParameters()
{
ValidAudiences = new string[]
{
"massrover.client",
},
ValidIssuers = new string[]
{
"massrover.authservice",
},
ValidateLifetime = true,
IssuerSigningKey = GetSignTokenSignKey()
};
var tokenHandler = new JwtSecurityTokenHandler();
tokenHandler.ValidateToken(token,tokenValidationParameters,out var
validatedToken);
return true;
}
```

The code uses the “TokenValidation Parameters” object, which includes token validation logic along with signing keys (retrieved from the sameprivate method used to issue the token). The “ValidateToken method” from JWT “Security Token Handler” validates the token using the constructed “Token Validation Parameters” object.

Note that the above code snippets are not production ready, and token flow in the production application requires software implementation along with TLS support. The above pieces of code explain the fundamentals of issuing and validating JWT tokens in ASP.NET Core. Once the JWT token flow is in place, consumers will send the token in each request to the API. In typical scenarios, tokens are sent in the Authorization header. Backend services receive the token from the HTTP request and validate and obtain information from them to coordinate the business logic requirements. In some cases, tokens do not contain any information other than an identifier, but backend services know to get the required information using this identifier. These kinds of tokens are known as reference tokens.

The next section will focus on how Azure API Management can be used to pre-authenticate requests that contain a JWT token.

15.7.5 Pre-Authentication in Azure API Management

Examples in presiding sections explained how the Azure API Management can process request and response information. Pre-authentication at the gateway is a good practice to make the API secure. This prevents requests from reaching the backend service of an API. Even though the requests do not reach the backend it is still strongly recommended to validate the token in the backend service and obtain the information stored there. Validate JWT policy is used for this pre-authentication step.

The code snippet [1] below shows a basic JWT validation policy implementation.

```
<validate-jwt
header-name="Authorization" failed-validation-httpcode="401 "
failed-validation-error-message="Unauthorized"
require-expiration-time="true"
require-scheme="scheme"
require-signed-tokens="true">
<audiences>
<audience>76d88779-d888-401f-8565-231aee385b14</audience>
</audiences>
<required-claims>
<claim name="massrover-role" match="any">
<value>admin</value>
<value>user</value>
</claim>
</required-claims>
<openid-config url=" https://login.microsoftonline.com/
common/.well-known/openid-configuration" />
</validate-jwt>
```

In the above snippet, the open id-config URL and validation comparison of basic claims are provided [1]. The code also checks a custom

claim called mass rover-role [1] and makes sure its presence in the JWT token and the value it can take either admin or user. Also, it should be noted that require-expiration-time is set to true. In order to pass this validation, the incoming JWT token should contain the exp claim. If the exp claim is not present, validation will fail. The Azure API Management JWT validation policy supports bothHS256 and RS256 signing algorithms. For HS256, they should be provided in the policy itself as a base64 encoded string, like below.

```
<issuer-signing-keys>
<key>base64 encoded key</key>
</issuer-signing-keys>
```

For RS256, the key must be provided via an OpenID configuration end point, as shown in the sample policy above. One can add more validation rules to this policy, including validating
custom claims.

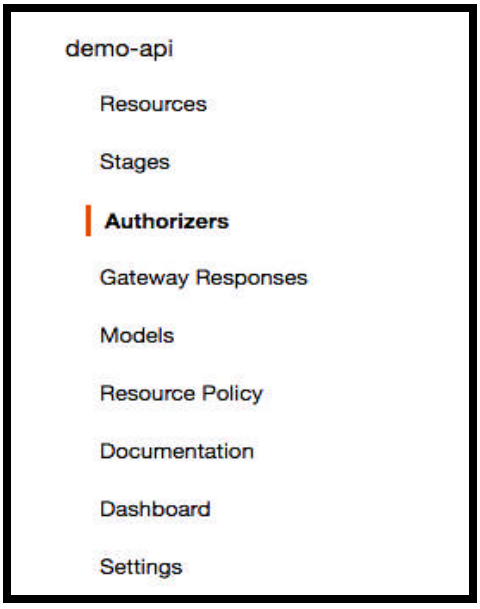
15.7.6 Authorizers in AWS API Gateway

Authorizers are set up in AWS API Gateway to authenticate incoming requests [1]. AWS API Gateway handles this with a Lambda function or AWS Cognito. Lambda is the server less platform of AWS, and Cognito is the AWS-based access control service.

To set up an authorizer follow the steps below:

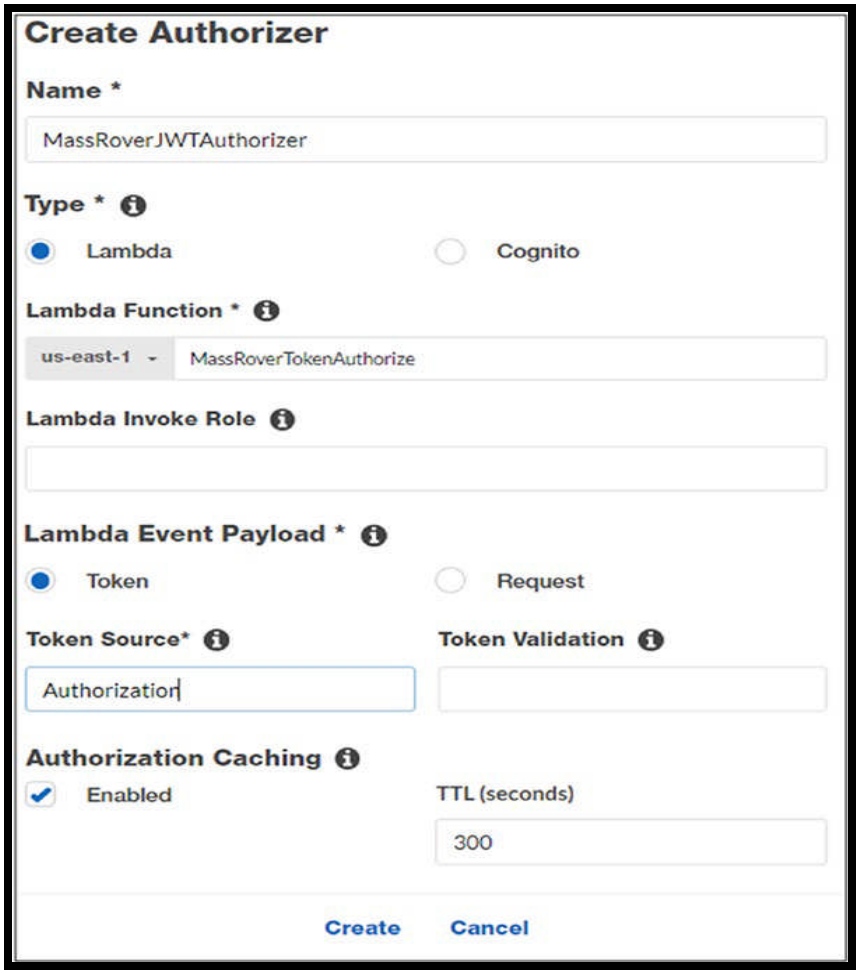
1. On API Gateway console left panel, choose the API and select ‘Authorizers, as shown in figure 15.18.

Figure 15.18 Creating Authorizers for a selected API source:https://jun711.github.io/aws/aws-api-gateway-access-control-with-iam-cognito-lambda-custom-authorizer/



2. In the Authorizers panel click Create New Authorizer tab as shown in figure 15.19.
3. Provide a name for the authorizer and select the type as Lambda. This is the name to which one can write the code to do the authentication validation logic.
4. Next, select the Lambda function, which has the authorizer logic. To do that,t here needs to bean existing Lambda function and implementation. If the function does not exist, provide a name for it in the textbox as shown in figure 15.19.This Lambda function will be deployed later. To learn to create and deploy Lambda functions the Reader can refer to chapter 6 of the book Practical API Architecture and Development with Azure and AWS by Thurupathan Vijayakumar.
5. Select a Token for the Lambda Event Payload; this ensures the token will be present in the specified header.
6. Select Request if the token is present in the event payload request body/header/query string with the specified value. Specify the corresponding value of the token in the Token Source.

Figure 15-19Create AWS API Gateway Authorizer source: *Practical API Architecture and Development with Azure and AWS-Thurupathan Vijayakuma*



The Lambda context will look for the token in the Authorization header. The rule to look for the token in headers comes from Lambda Event Payload type, and the value comes from Token Source. The same rule will apply when the Request is selected as the Lambda Event Payload. The lookup for the token will be happening in a range of places like the headers, query string parameters, stage variables, and context parameters for the specified Token Source key.

For doing Token Validation a Regular Expression validator can be used in the Token Validation section. This will prevent errors messages from coming up for Lambda having tokens that are not in the right format. In the figure 15.19, Authorization caching when enabled indicates whether to cache the authorization policy document or not. Authorization policy documents are generated by the authorizer for the specified token. The Authorization policy documents dictate whether AWS API Gateway has access to specific AWS resources or not. The policy document is produced based on the logic in the authorizer Lambda function.

The policy documents control access to AWS resources. The authorizer Lambda should have the token validation logic for the application business logic. Once the token validation is performed, the Lambda will construct the policy document and return it to AWS API Gateway, which will then cache the document for the determined period if caching is enabled.

The Lambda authorizer created above will be implementing in the code below [1] which contains the logic for validating the token and generating the policy document.

```
public class Function
{
public Policy Function Handler (API Gateway Custom Authorizer
Request auth Request,
ILambda Context context)
{
var token = authRequest.AuthorizationToken;
Policy policy;
if (ValidateToken(token))
{
var statement = new Statement(Statement.
StatementEffect.Allow);
var policyStatements = new List<Statement> {
statement };
policy = new Policy("TokenValidationPassed",
policyStatements);
}
else
{
var statement = new Statement(Statement.
```

```
StatementEffect.Deny);
var policyStatements = new List<Statement> {
statement };
policy = new Policy("TokenValidationFailed",
policyStatements);
}
return policy;
}
private bool ValidateToken(string token)
{
// JWT token validation here
return true;
}
}
```

The policy document written in the code above indicates whether access to AWSresources is allowed or not. It does not contain any specific access policy to any specific AWSresources.

SUMMARY

This chapter explained the concept related to the management of API Gateways in public cloud.

It explained the following concepts:

- Endpoint mapping with different types of mappings between backend service endpoints and API gateway interface endpoints.
- Implementing two popular API management services in public cloud- Microsoft Azure and the Amazon based AWS.
- Implementing security on a public cloud through various security mechanisms like Request-based security and Authentication and Authorization.
- Implementation of these mechanisms in Azure and AWS.

BIBLIOGRAPHY:

1.https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio
2.https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/
3.Hubspot

REFERENCES

1. Practical API Architecture and Development with Azure and AWS- Thurupathan Vijayakuma ,*ISBN-13 (pbk): 978-1-4842-3554-6,ISBN-*

13 (electronic): 978-1-4842-3555-3,https://doi.org/10.1007/978-1-4842-3555-3 ,Library of Congress Control Number: 2018946567

2. 1.**https://www.wrike.com/blog/application-programming-interface-api-explained/#:~:text=Other%20examples%20of%20APIs%20that,constellation%20data%20for%20public%20use.** *By Andrew Slate, May 31, 2019*
3. ***https://blog.restcase.com/restful-api-authentication-basics/***

MODEL QUESTIONS

Multiple Choice Questions

1.APIs perform value-added functions like_____.
A) storage B) caching C) cookies D) none

2. _____ is a function of an API gateway endpoint.
A) Verifying API keysB)Using Explicit Parameters C) Query string parameter D) none
3. Azure _____portal is the administrative interface where one can set up their API program.
A) Publisher B) DeveloperC)Gateway endpoint D) none

4. _____ are a great way to strength security in API.
A) passwords B) keys C) Tokens D) none

5. In an API usage plans the rate limit can be set via the _____ parameter.
A) HTTP status codes B) bandwidthC) Throttling D) none

Answers to Multiple Choice Questions

1. B 2.B 3.A 4.C 5.C

Theory Question

1. State and explain some of the most common ways to strengthen API security
2. Write a short note on Authentication and authorization?
3. What is Request-based security in AWS API Gateway and what are the various ways it can be implemented?
4. Write a short note on API Keys?
5. Write the steps for setting up Authorizers in AWS API Gateway?

❖❖❖❖