

Cloud-Native Development



ROITRNING
MAXIMIZE YOUR TRAINING INVESTMENT™

The following course materials are copyright protected materials. They may not be reproduced or distributed and may only be used by students attending the *Cloud-Native Development* course.

Welcome!

- ROI leads the industry in designing and delivering customized technology and management training solutions
- Meet your instructor
 - Name
 - Background
 - Contact info
- Let's get started!



Intended Audience

- Software architects designing applications for the cloud
- Developers working on modern, cloud-based, microservice applications
- IT and DevOps professionals who want to leverage modern automation tools

Course Prerequisites

To get the most out of this course, you should have:

- Basic understanding of cloud computing
- Experience administering or deploying applications to Linux or Windows
- Experience programming web applications or services
- Basic understanding of data storage

Course Objectives

In this course, you will learn how to:

- Design, architect, build, and deploy cloud-native applications
- Choose the right services for public, private, and hybrid cloud deployments
- Program applications using cloud-native best practices
- Optimize cloud resources by leveraging microservice architectures
- Containerize applications using Docker
- Orchestrate container deployment using Kubernetes
- Simplify microservice deployments using PaaS and serverless environments
- Leverage cloud-based data storage services
- Automate builds and deployments using modern DevOps tools

Course Contents

Chapter 1	Cloud Computing Overview
Chapter 2	Cloud-Native Development
Chapter 3	Microservices
Chapter 4	Application Lifecycle Management
Chapter 5	Docker
Chapter 6	Kubernetes
Chapter 7	Cloud Data Services
Chapter 8	Platform as a Service
Chapter 9	DevOps Automation (CI/CD)

Course Schedule

- Start of class: _____
- Break: _____
- Lunch: _____
- Resume: _____
- Break: _____
- Class ends: _____

Student Introductions

Please introduce yourself stating:

- Name
- Position or role
- Current project you are working on
 - What is the current phase or stage of the project?
- Expectations or a question you'd like answered during this class



End of Course Assessment

- This course contains an end of course assessment
 - Will be administered on the last day
- Completing this assessment is required



Case Study

Case Study: Events Feed

- Your instructor will put you in teams to work on a case study
 - Your team will need to demo your work
- You will work on an application that the company can use to announce events, news, and important messages
- Possible features include:
 - Anyone should be allowed to post events
 - Users should be able to “Like” events
 - Users should be able to comment on events
 - Show a map of the event location
 - Users have to sign up and sign in to use the service
 - Any other features you think are important

Case Study Requirements

- The implementation of your case study has to follow the best practices taught in the class
 - Use a microservice architecture
 - Follow Twelve-Factor App best practices
 - Automate builds, testing, and code analysis in a CI/CD pipeline
 - Use cloud services
 - Deploy to a cloud provider using Kubernetes
 - Etc.
- Here's a similar program: <https://hiplocal.kwikstart.net/>

Case Study Implementation

- A starting point is provided in Node.js
 - The application is only partially implemented
 - You will add functionality during the course
 - Or implement in another language
 - Java, Python, Node.js, Go, .NET, etc.

Case Study Deployment

- In this course, we will use the cloud to deploy the case study
 - The principles learned can be applied to any cloud provider
 - Amazon Web Services (AWS), Google Cloud, Microsoft Azure, etc.
 - This course focuses on open-source and/or cloud agnostic tools
 - Git
 - Docker
 - Kubernetes
 - CI/CD
 - Etc.
- You will be provided with a cloud account for use during the class
 - Not available for use after class

Cloud Accounts

- Most cloud providers offer a free trial
 - AWS provides free quotas per month for the first year
 - Google Cloud provides a \$300 credit
 - Microsoft Azure provides free quotas per month for the first year plus \$200 credit for the first month
- If you would like to create your own free trial account, perform one of the following (for homework):
 - AWS
 - <https://aws.amazon.com/free>
 - Google Cloud
 - <https://cloud.google.com/free>
 - Azure
 - <http://azure.microsoft.com/free>

Activity: Accessing the Class Environment

Access the cloud environment provided for the class:

- Log in using credentials provided by your instructor
- Configure the environment as required for the class

Qwiklabs

- Qwiklabs credits will be provided to allow you to gain more hands-on experience
 - These credits are for you to use outside of class hours to gain more knowledge on a topic
 - Generally, Qwiklabs will not be done in class
- You will need to create a free Qwiklabs account:
 - Go to <https://www.qwiklabs.com/>
 - Click the **Join** button
 - Follow the prompts to create new account
- Google Cloud labs can be found at <https://www.qwiklabs.com/>
- AWS labs can be found at <https://amazon.qwiklabs.com/>



Cloud Computing Overview

Chapter Objectives

In this chapter, you will:

- Define Cloud Computing
- Explore characteristics of cloud computing and cloud platforms
- Identify cloud service and deployment models

Chapter Concepts

Defining Cloud Computing

Cloud Deployment Models

Cloud Platforms

Cloud Service Models

What Is Cloud Computing?

- Cloud computing often means different things to different people
 - Depending on their experience/context
 - What is cloud computing to you?
-  How do you define it?
-

What Is Cloud Computing? (continued)

- Many definitions exist for cloud computing:

“A style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies.”

—Gartner Report

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

—www.nist.gov

What Is Cloud Computing? (continued)

- Main goal of cloud computing is to allow IT resources to be delivered and consumed as a service
 - The same way services like electricity, telephone, etc. are delivered and consumed
 - Allows organizations to focus on their core business
 - And not become experts in deploying, managing, maintaining IT infrastructures
- The simplest definition of cloud computing:



Cloud Computing = IT as a Utility

Key Cloud Characteristics

- Massive scalability and rapid elasticity
 - To the point that resources appear to be unlimited
 - Including compute, storage, and network capacity
- Measured service, or “pay as you go” only for what is needed
- Pooling of shared resources—networks, servers, storage, applications
 - Requires a **multi-tenancy** model
- On-demand, self-service
 - Increase or decrease resources as needed
 - Provision without requiring human interaction with service provider
- Capabilities accessed over the network
- Capital expenditure is often converted to operational expenditure

Cloud Computing Actors

Actor	Definition
Consumer	A person or organization that maintains a business relationship with, and uses service from, <i>Cloud Providers</i>
Provider	A person, organization, or entity responsible for making a service available to interested parties
Advisor	A party that can conduct independent assessment of cloud services, information system operations, performance, and security of the cloud implementation
Broker	An entity that manages the use, performance, and delivery of cloud services, and negotiates relationships between <i>Cloud Providers</i> and <i>Cloud Consumers</i>
Carrier	An intermediary that provides connectivity and transport of cloud services from <i>Cloud Providers</i> to <i>Cloud Consumers</i>



Where do you fit in? _____

Chapter Concepts

Defining Cloud Computing

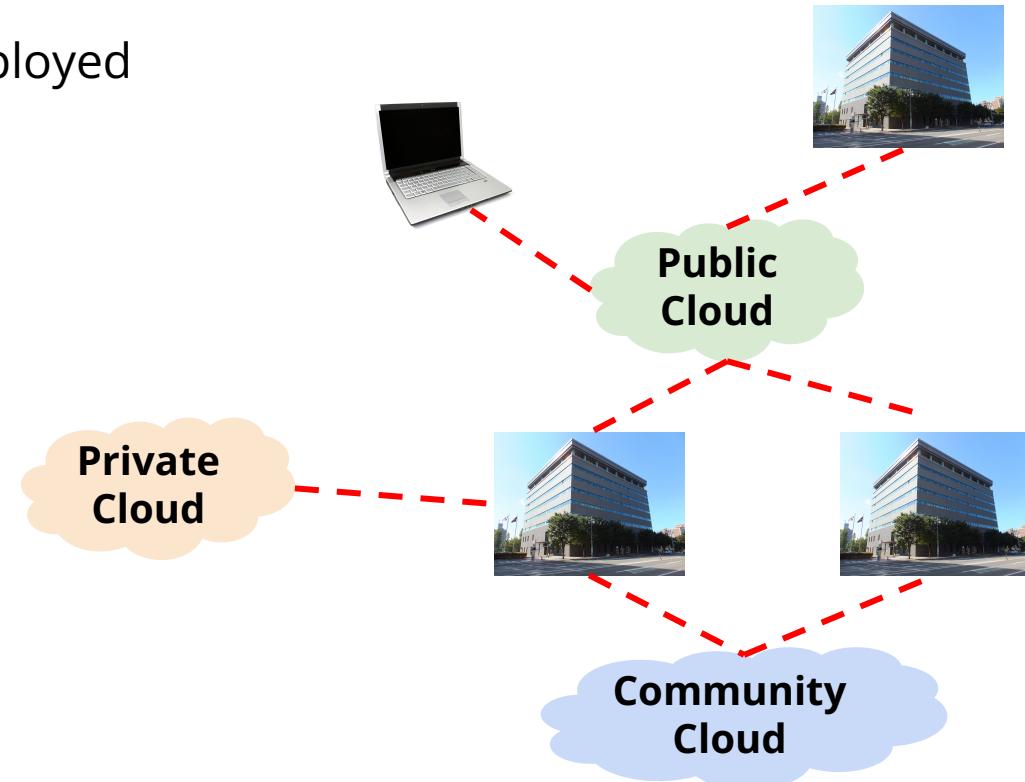
Cloud Deployment Models

Cloud Platforms

Cloud Service Models

Deployment Models

- Cloud computing can be deployed in several different ways
 - Public
 - Private
 - On site
 - Hosted
 - Community
 - Hybrid
 - Multi



Chapter Concepts

Defining Cloud Computing

Cloud Deployment Models

Cloud Platforms

Cloud Service Models

Cloud Platforms

- There are many different cloud platforms
 - Cloud providers are companies that offer cloud platforms for development, management, and deployment of applications
- The top five public cloud providers:
 - **Amazon Web Services (AWS)**
 - Publicly available since 2006
 - **Microsoft Azure**
 - **Google Cloud**
 - IBM Cloud
 - Alibaba Cloud
- AWS, Azure, and Google Cloud are usually considered the major players

Chapter Concepts

Defining Cloud Computing

Cloud Deployment Models

Cloud Platforms

Cloud Service Models

IT Ops/DevOps/NoOps



- IT Ops (or traditional ops or TechOps)
 - Term referring to the management of traditional IT systems
 - Often refers to three areas: network infrastructure, computer operations, and device management
- DevOps
 - An approach to unify software development, testing, and operation
 - Aims to speed the time to market through automation and agile development
- NoOps or “Serverless”
 - All IT operations are managed by the provider
- The cloud can provide all of these
 - And the ability to easily switch between them

Cloud Service Models

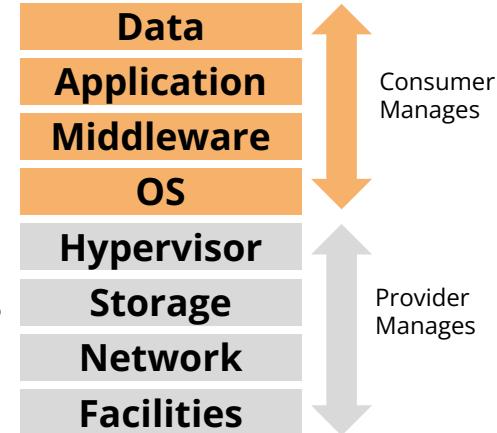
- Cloud services are commonly divided into three categories
 - Infrastructure as a Service (IaaS)
 - Platform as a Service (PaaS)
 - Software as a Service (SaaS)

On-Premises	IaaS	PaaS	SaaS
Data	Data	Data	Data
Application	Application	Application	Application
Middleware	Middleware	Middleware	Middleware
OS	OS	OS	OS
Hypervisor	Hypervisor	Hypervisor	Hypervisor
Storage	Storage	Storage	Storage
Network	Network	Network	Network
Facilities	Facilities	Facilities	Facilities

 = Consumer Managed
 = Provider Managed

Infrastructure as a Service (IaaS)

- Delivers core infrastructure as a service
 - Networks, storage, servers, etc.
- IaaS provides the most flexible cloud solution
 - Build applications on top of this infrastructure
 - Not much different from administering local servers
 - Can configure and install any software you want
- But consumers are responsible for maintaining it
 - Cloud provider does not maintain, backup, patch, update the software
- Often the easiest path onto the cloud
 - Migrate on-premises VMs to VMs hosted in the cloud

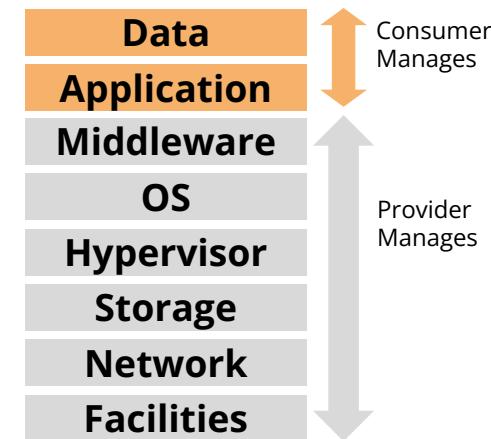


Qwiklabs: Creating a VM in the Cloud

- AWS
 - [Introduction to Amazon EC2](#)
- Google Cloud
 - [Creating a Virtual Machine](#)
 - [Compute Engine: Qwik Start - Windows](#) (Windows)

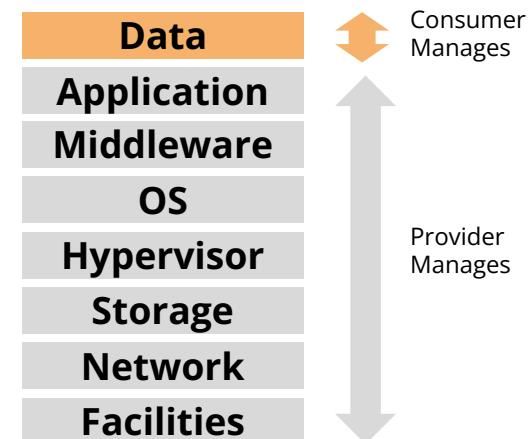
Platform as a Service (PaaS)

- Delivery of a hosted application environment as a service
 - Provides resources required to deploy and execute an application
 - Including auto-scaling, load balancing, health checks, etc.
- No need to buy or maintain resources required to execute the software
 - Simply deploy your application code into the environment
 - Enables organizations to concentrate on area of expertise
 - And not worry about managing IT infrastructure
 - Downside is code must conform to rules of the platform
 - Might require re-work for existing applications
- Many combinations of services by different PaaS providers



Software as a Service (SaaS)

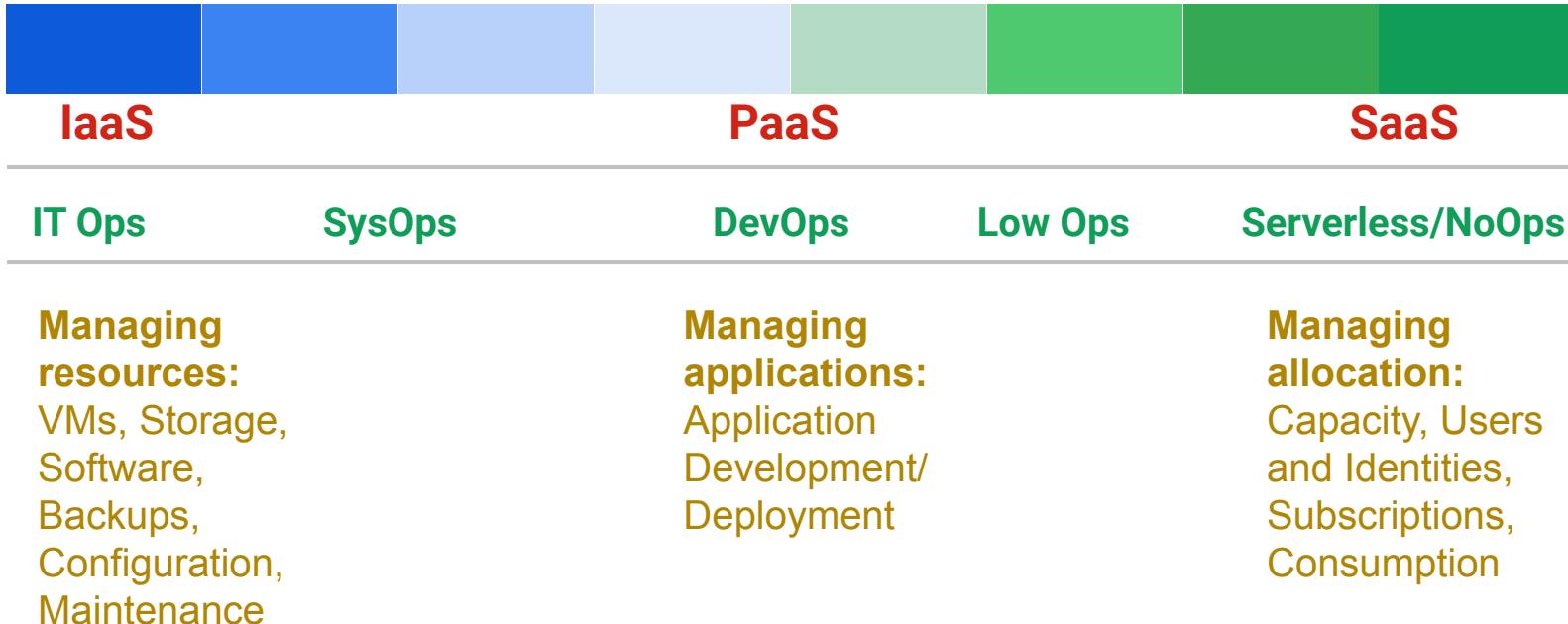
- Model where provider licenses application use as a service on demand
 - No need to manually deploy or manage infrastructure in the cloud
 - No need to install software on the client computer
 - Provider supplies the entire infrastructure and software application
- Many traditional software applications have been rewritten as SaaS
 - Email, office suites, accounting software, productivity tools, CRM
 - Accessed using a web browser



SaaS - Serverless Computing

- SaaS can also be used to deliver cloud services
 - Some sophisticated cloud services are consumed as a SaaS NoOps solution
 - Sometimes called *serverless* computing
 - No need to manually provision or manage servers
 - There are still servers running, but the provider dynamically manages the allocation of machine resources
- Some examples of serverless cloud services:
 - Data warehouses
 - Data analytics
 - Machine learning libraries

Cloud Service Models (continued)



Activity: Planning the Case Study Architecture

- In your groups, analyze the case study and plan which service models you may be able to leverage
- Keep your notes in a Google Slides document
 - You will continue to add to this document throughout the course
- From a high-level perspective, start by listing the kinds of services you think the application will need; i.e.:
 - From IaaS to SaaS – list the models you think that your app will need
 - Databases
 - File Storage
 - Etc.

Chapter Summary

In this chapter, you have:

- Defined Cloud Computing
- Explored characteristics of cloud computing and cloud platforms
- Identified cloud service and deployment models

Quiz

Which is an advantage of cloud computing?

- A. Customer resource isolation
- B. Elastic scalability
- C. Pay upfront, predictable capital expense
- D. All of the above

Quiz

Amazon Web Services is an example of which cloud computing deployment model?

- A. Public Cloud
- B. Private Cloud
- C. Hybrid Cloud
- D. Community Cloud

Quiz

If you created a network on AWS and deployed your applications to virtual machines in EC2, that would be an example of which cloud service model?

- A. Infrastructure as a Service (IaaS)
- B. Platform as a Service (PaaS)
- C. Software as a Service (SaaS)
- D. Database as a Service (DaaS)



ROITRAINING
MAXIMIZE YOUR TRAINING INVESTMENT™

Cloud-Native Development

Chapter Objectives

In this chapter, you will:

- Consider the need for cloud-native development
- Plan projects using agile best practices
- Define automation and DevOps required for cloud-native development

Chapter Concepts

Cloud-Native Computing

Agile Practices

Automation and DevOps

What Is Cloud-Native Computing?

- Cloud-native applications take advantage of cloud computing concepts
 - PaaS
 - Microservices
 - DevOps
 - Containers
 - Automation
 - CI/CD
 - Agile methodology
 - Multi cloud
 - Etc.
- To create applications that are more flexible, maintainable, scalable, reliable, ...

What Is Cloud-Native Computing? (continued)

- Cloud native applications do not need to be deployed to the public cloud
 - Term is used to describe how applications are built, deployed, and managed
 - Not necessarily where they are deployed

Traditional Development vs. Cloud Native

	Traditional	Cloud Native
Development Methodology	Waterfall	DevOps/Agile
Teams	Separate development, operations, security, and QA teams	DevOps/SecDevOps
Delivery timeframe	Long	Very short and continuous
Architecture	Monolithic, tightly coupled	Loosely coupled, Microservices, APIs
Infrastructure	Server (VM) based, Vertical scaling	Container-based, Horizontal scaling

Chapter Concepts

Cloud-Native Development

Agile Practices

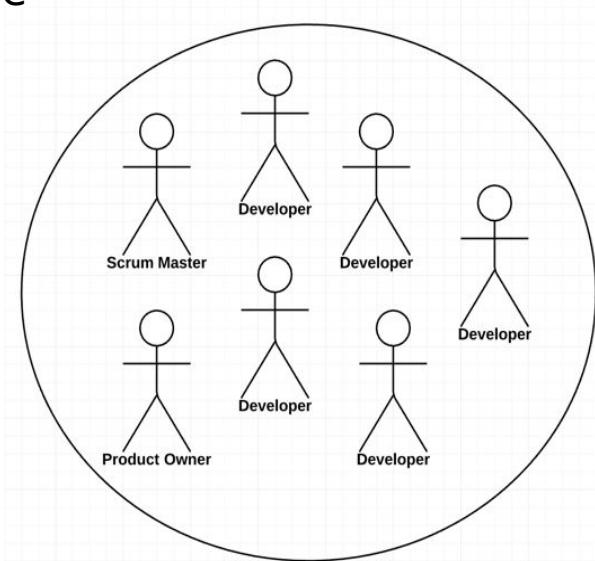
Automation and DevOps

Agile Management Practices

- Short iterations
- Self-organizing teams
- Feature- and value-driven development
- Regular reviews and retrospectives
- Others?
 - _____
 - _____
 - _____

Agile Teams

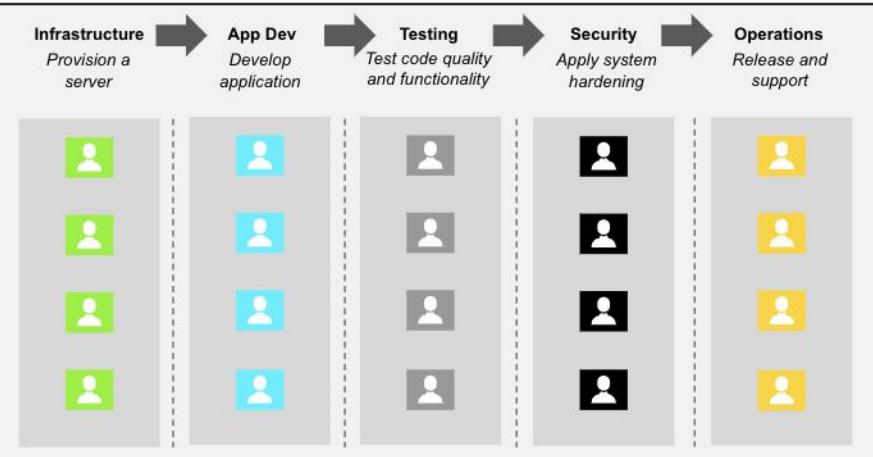
- Consist of a Scrum Master, Product Owner, and a collection of Developers
 - Developers work in a variety of roles to get the software done
- Agile teams are cross-functional and collaborative
- Agile teams require:
 - Cross training
 - People willing to learn new things
 - People with multiple skill sets
- Agile teams maximize:
 - Communication
 - Collaboration
 - Productivity



Traditional Teams Are Siloed, DevOps Teams Need to Be Collaborative

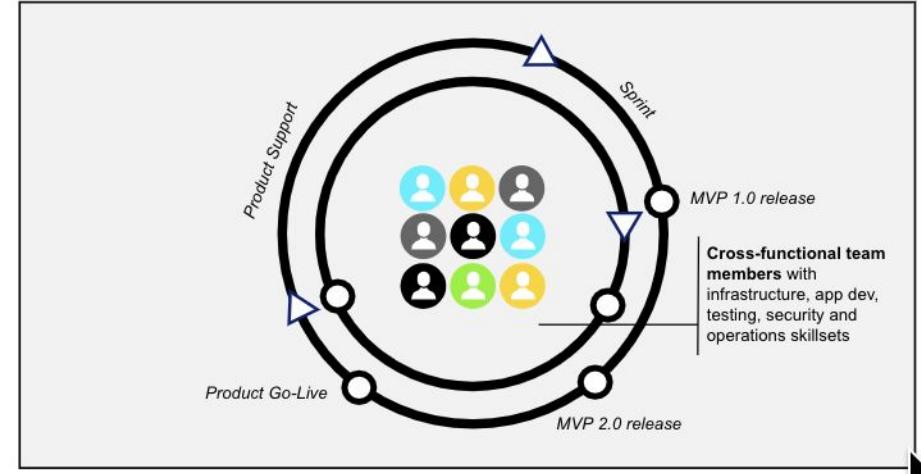
Traditional Team Structures

Teams organized by skill sets, focused on stability instead of innovation and change



DevOps-Enabled Teams

Teams become cross-functional, able to deliver end-to-end



Feature-Driven Development

- An application is decomposed into small features
 - Each feature is known as a user story
- User stories must be small enough in scope to fit into an iteration
 - If a story is too big to fit in an iteration, it must be broken down further
- User stories must describe features that a user would be interested in
 - Each must provide some amount of business value
- All scheduling, planning, and prioritization is focused on stories (*features*)
 - Hence, the term feature-driven development
 - Any activity that doesn't help get a story completed would be considered a waste

User Stories

- Users stories describe a feature of your software in one sentence
 - Must be written from the user's point of view
- User stories contain:
 - A title
 - The story
 - A priority
 - An estimate of effort
 - An estimate of value
- Use a template to maintain consistency:
 - As a (*user role*), I want to (*describe the goal*), so (*describe why this is important to the user*).

Prioritization

- In any program, some features are more important than others
 - Some are so important that without them there is no software
 - Others can wait until after the first version is deployed
 - Other features would be nice to have if the team has time
- Make prioritization easy!
 - Mark every story as **High**, **Medium**, or **Low** priority
 - Don't worry about medium-priority features until all the high-priority features are finished

Estimating Effort

- Estimate the work required to complete each story relative to the others
- Use an abstract scale that increases at an increasing rate
 - Fibonacci series: 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
- Do not estimate in hours or days
 - Time-based estimations are too hard and too inaccurate
- Tips:
 - To get started, find the easiest, high-priority story and give it an estimate of 8
 - Estimate every other story relative to that one
 - Most stories should be in the middle (8, 13, 21)

Business Value Estimation

- Similar to estimating effort, but you are estimating the business value provided by the feature
- Use an abstract scale that increases at an increasing rate
 - For example: 50, 100, 250, 500, 1000, 1500, 2000, 3000, 5000, 8000, ...
- Do not estimate in dollars
 - Estimating in money is too hard to quantify
- Tips:
 - To get started, find the most valuable, most important story and give it an estimate of 5000
 - Estimate every other story relative to that one

User Story Example

Subscribe to Event

As an event participant, I want to subscribe to events that I plan on attending, so I am kept up to date on any changes.

Priority: H

Work Estimate: 13

Value Estimate: 3000

Value-Driven Development

- Value-driven development is simple and obvious
 - Work on higher priority features first
 - Of the high-priority features, work on the ones with higher value first
 - If features have the same business value, work on the easier ones first

Planning and Tracking Work

- Product backlog
 - The prioritized list of features (user stories) planned for the entire product
 - Sort by priority, value, and effort
- Sprint (iteration) backlog
 - The features (user stories) the team is working on in the current Sprint
 - User stories are broken into tasks
 - Each task is signed up for by a developer
- Burndown chart
 - A chart that shows the amount of work left
 - Used to track team progress

Activity: Analyzing the Case Study Requirements

- In your groups, write user stories for the course case study
 - Try to create at least one story for each member in your group
- Do this in the Google Slides document started earlier
 - Create one slide per story
- Each story should have the following:
 - Title
 - Description (use the format: *As a (role), I want to..., so ...*)
 - Priority (H,M,L)
 - Effort estimate (1, 2, 3, 5, 8, 13, 21, 34, 55)
 - Value estimate (50, 100, 250, 500, 1000, 1500, 2000, 3000, 5000)

Chapter Concepts

Cloud-Native Development

Agile Practices

Automation and DevOps

Automation

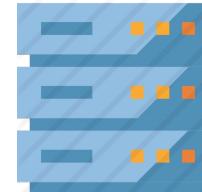
- Software is complex and only getting more so
 - Humans need help automating repetitive, time-consuming tasks
 - Computers are really fast and never make mistakes
 - Computers enjoy repetitive tasks!
- Automation can help with:
 - Tests
 - Builds
 - Deployment
 - Configuration
 - Etc.

Source and Version Control

- Manage code and versions using a modern version control system
 - Distributed version control like Git allows for multiple sources
 - Use branches to manage versions
 - Monitor branches and respond to changes with automated builds

Infrastructure as Code (IaC)

- Managing and provisioning resources through definition files
 - Creating standard, repeatable processes for building, modifying, and destroying entire environments
- Examples:
 - Terraform
 - AWS CloudFormation
 - Google Deployment Manager
 - Azure Resource Manager



Continuous Integration

- The goal of continuous integration is to make deployment of new software and new versions non-events
 - Test environments should duplicate production environments
 - Deployments into test environments should be completely automated
 - Deployments happen many times per day
- When developers check in their code:
 - An automated build should compile the code
 - The automated tests should run
 - Creation of dependencies like databases should be scripted
 - The software is deployed for testing
- Deploying into production should be as simple as a configuration change

Continuous Delivery

- The next step after continuous integration
 - Automate the build to the point it can determine when all quality standards have been met, and when they have deploy to production
- Requires a deployment pipeline be set up
 - The pipeline chooses to deploy to a test environment or a staging environment
- Staging environment exists on production platform
 - Multiple versions can be deployed simultaneously
 - Easy to switch from one version to the other
 - If a mistake is made, roll back to prior version

Automated Builds

- A build consists of:
 - Downloading application dependencies
 - Copying source code
 - Compiling source code
 - Setting environment variables
 - Creating a deployment package
- Deployment packages can be:
 - Docker images
 - WAR, JAR, Zip files, etc.
- As much as possible the process of executing a build should be scripted
 - Tools like Maven, NPM, Docker make automating builds easier

Automated Testing

- Testing needs to be done as much as possible without human intervention
- Use automated testing tools to test as much as possible
 - xUnit for unit and integration testing
 - Many tools exist for blackbox security, system and performance testing
 - Selenium can be used to script and automate end-to-end testing
- If most of your testing is automated, human testers can focus on usability, not finding bugs

Automated Code Analysis

- Testing helps find the bugs, code analysis helps ensure code is easy to understand, efficient, and maintainable
- Automated code analysis tools can detect:
 - Poor naming and coding conventions
 - Unused variables and functions
 - Inefficient coding practices
 - Potential security flaws (SQL Injection flaws, for example)
- SonarQube is a common tool
 - See: <https://www.sonarqube.org/>

CI/CD Pipelines

- CI/CD pipelines perform the operations required to deploy the software to a test or production environment
 - Start the pipeline
 - Run the automated build
 - Run the automated tests
 - Run the code analysis
 - Etc.
- CI/CD pipelines collect data and metrics along the way
 - Allows reports to be run to determine the quality at each step

Triggering a Pipeline

- CICD pipelines are typically triggered when software is checked into the source code control system
 - The pipeline watches source code for changes
 - Or the source control system executes a webhook on check in

Activity: Designing a CI/CD Pipeline

- In your groups, create a diagram that depicts how your CI/CD pipeline should work

Chapter Summary

In this chapter, you have:

- Considered the need for cloud-native development
- Planned projects using agile best practices
- Defined automation and DevOps required for cloud-native development

Quiz

Which would be considered a best practice of cloud-native development?

- A. Deploy apps using containers
- B. Design for microservices
- C. Manage code using a source control system like Git
- D. All of the above

Quiz

What would be the most likely way to trigger a continuous integration pipeline?

- A. Use a command line function
- B. Execute the pipeline within Jenkins by selecting it
- C. Monitor for changes in a Git repository
- D. Run a scheduled job using cron

Quiz

What is an advantage of using Infrastructure as Code tools like Terraform or AWS CloudFormation?

- A. Faster provision of cloud resources
- B. Easier to repeat identical cloud environments
- C. Can store resource templates using source control
- D. All of the above



Microservices

Chapter Objectives

In this chapter, you will:

- Design systems using a microservice style architecture
- Follow Twelve-Factor App best practices
- Understand microservice communication protocols

Chapter Concepts

Introduction to Microservices

Twelve-Factor App

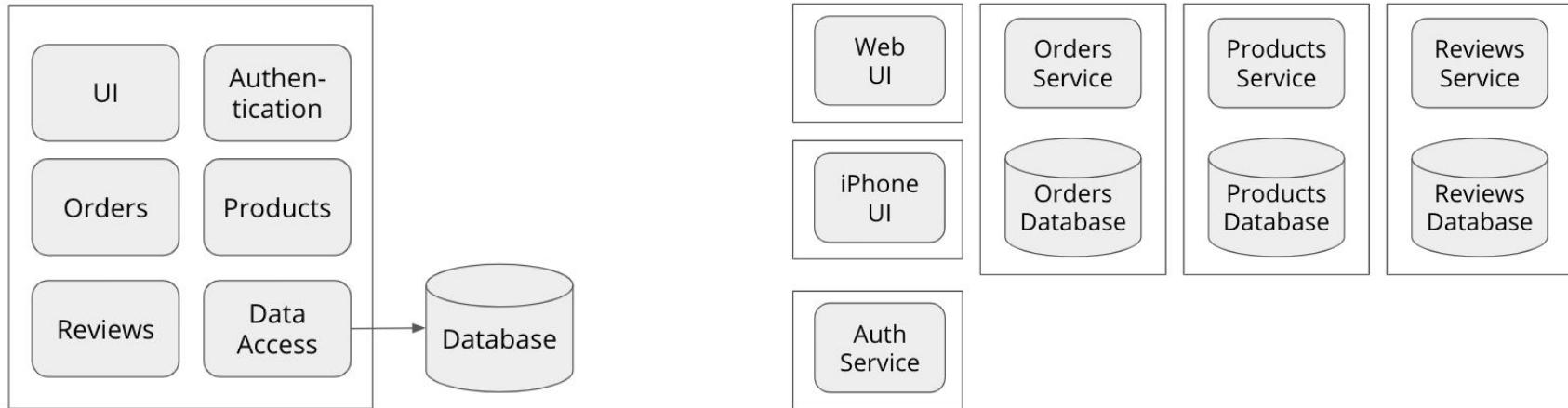
Microservice Architecture

Microservice Architecture

- Divide a large program into a number of smaller, independent services
 - Each service is programmed, deployed, and run separately
- Advantages of microservices include:
 - Reduced risk when deploying new versions
 - Services scale independently to optimize use of infrastructure
 - Easier to innovate and add new features
 - Can use different languages and frameworks for different services

Monolithic vs. Microservice Architecture

- Monolithic applications implement all features in a single code base
 - Single database for all data
- Microservices break large programs into a number of smaller services
 - Each service manages its own data

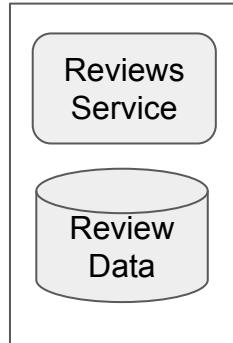


Recognizing Microservice Boundaries

- Defining service boundaries is a matter of decomposition
 - Deciding what pieces of the larger application should be separated
- Typically done by features to minimize dependencies across services
 - Reviews service, Order Processing service, Product Catalog service, etc.
- Sometimes services are organized by architectural layer
 - Web, iPhone, and Android user interfaces
 - Services that provide access to shared data
- Some services provide shared behavior to the others
 - Authentication service, for example

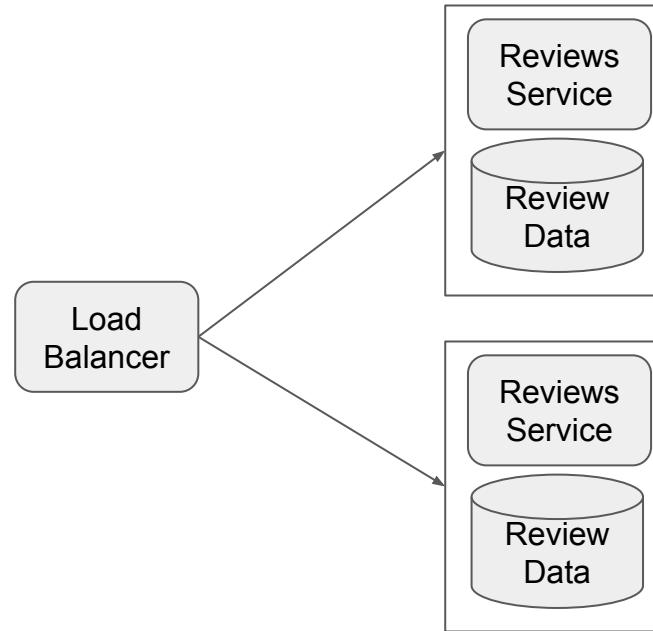
Stateful vs. Stateless Services

- Stateful services manage stored data over time
 - Harder to scale
 - Harder to upgrade
 - Need to back up
- Stateless services get their data elsewhere
 - Easy to scale by adding instances
 - Easy to migrate to new versions
 - Easy to administer



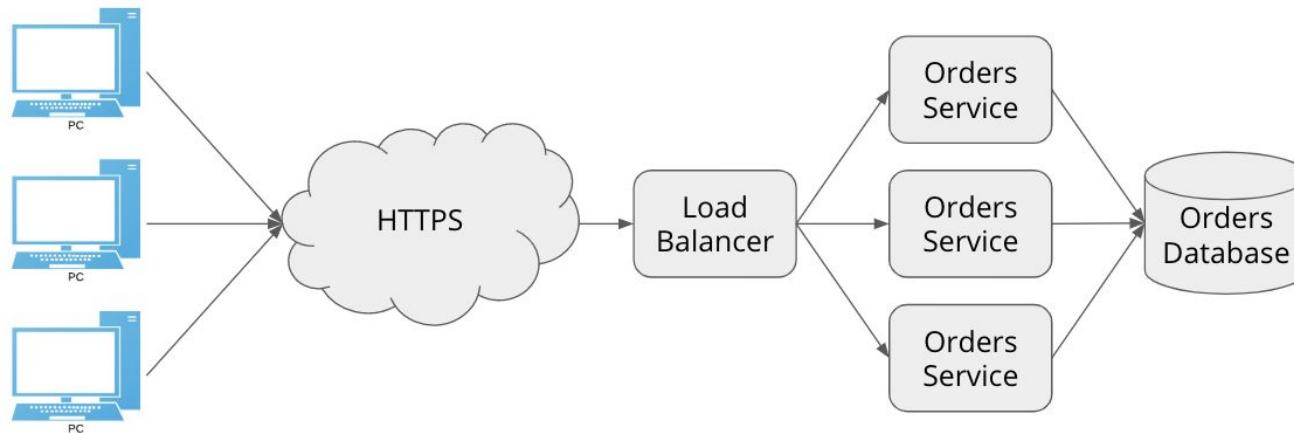
What Happens If a Stateful Service Scales?

- The data returned will depend on the instance assigned by the load balancer
 - Some systems solve this problem by using sticky sessions



Stateless Services

- Design services so state information is stored in a database or on the client
 - Don't store session information in memory on the service instance
- Service instances need to be able to scale up or down to meet demand
- Sticky sessions should not be required when configuring a load balancer



Asynchronous Operations

- Client requests to services should be made on a background thread
 - UI remains responsive while waiting for response
- Message queues can be used to decouple service requests
 - Use Pub/Sub service in Google Cloud to send messages to services
- Use event-driven services like Cloud Functions to simplify asynchronous operations

Logging and Monitoring

- Log every request to your service—both successful requests and errors
 - Allows application usage to be analyzed
 - Use a central log repository
- Monitor key metrics to ensure service performance
 - Uptime, health, latency, reads, writes, response time, CPU utilization, etc.

Managing Databases

- Having a single, main database for all services is considered poor design when implementing microservices
 - Each microservice should be responsible for its own data
 - Only one microservice should read and write into a datastore
 - Sharing data with other services should be done via the microservice responsible for the data
- Choose the database technology most appropriate to each service
 - Blob stores for files
 - NoSQL databases when adequate
 - Relational databases when strong transactions and SQL are desired

Activity: Architecting Microservice Applications

- Draw an initial microservice design of your case study
 - Depict each service along with its data source

Chapter Concepts

Introduction to Microservices

Twelve-Factor App

Microservice Architecture

Twelve-Factor App

- A set of best practices for designing microservice applications
- See: <https://12factor.net/>



The screenshot shows the homepage of the [Twelve-Factor App](https://12factor.net/) website. The header features a stylized diamond logo above the title "THE TWELVE-FACTOR APP". Below the title is a large, thin horizontal bar. The main content area has a light gray background and contains the following sections:

- INTRODUCTION**: A paragraph explaining that software is commonly delivered as a service (web apps or SaaS) and introducing the twelve-factor methodology.
- A bulleted list of十二-factor principles:
 - Use declarative formats for setup automation, to minimize time and cost for new developers joining the project;
 - Have a clean contract with the underlying operating system, offering maximum portability between execution environments;
 - Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
 - Minimize divergence between development and production, enabling continuous deployment for maximum agility;
 - And can scale up without significant changes to tooling, architecture, or development practices.
- A note at the bottom stating: "The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc.)."

The Twelve Factors

1.	Codebase One codebase tracked in revision control, many deploys	<ul style="list-style-type: none">• Use a version control system like Git• Each app has one code base and vice versa
2.	Dependencies Explicitly declare and isolate dependencies	<ul style="list-style-type: none">• Use a package manager like Maven, Pip, NPM to install dependencies• Declare dependencies in your code base
3.	Config Store config in the environment	<ul style="list-style-type: none">• Don't put secrets, connection strings, endpoints, etc. in source code• Store those as environment variables
4.	Backing services Treat backing services as attached resources	<ul style="list-style-type: none">• Databases, caches, queues, and other services are accessed via URLs• Should be easy to swap one implementation for another

The Twelve Factors (continued)

5.	<p>Build, release, run Strictly separate build and run stages</p>	<ul style="list-style-type: none">• Build creates a deployment package from the source code• Release combines the deployment with configuration in the runtime environment• Run executes the application
6.	<p>Processes Execute the app as one or more stateless processes</p>	<ul style="list-style-type: none">• Apps run in one or more processes• Each instance of the app gets its data from a separate database service
7.	<p>Port binding Export services via port binding</p>	<ul style="list-style-type: none">• Apps are self-contained and expose a port and protocol internally• Apps are not injected into a separate server like Apache
8.	<p>Concurrency Scale out via the process model</p>	<ul style="list-style-type: none">• Since apps are self-contained and run in separate process, they scale easily by adding instances

The Twelve Factors (continued)

9.	Disposability Maximize robustness with fast startup and graceful shutdown	<ul style="list-style-type: none">• Apps instances should scale quickly when needed• If an instance is not needed, you should be able to turn it off with no side effects
10.	Dev/prod parity Keep development, staging, and production as similar as possible	<ul style="list-style-type: none">• Container systems like Docker makes this easier• Leverage infrastructure as code to make environments easy to create
11.	Logs Treat logs as event streams	<ul style="list-style-type: none">• Write log messages to standard out and aggregate all logs to a single source
12.	Admin processes Run admin/management tasks as one-off processes	<ul style="list-style-type: none">• Run admin tasks as a shell command on the execution environment• Admin tasks shouldn't be a part of the application

Chapter Concepts

Introduction to Microservices

Twelve-Factor App

Microservice Architecture

Designing Loosely-Coupled Services

- Clients should not need to know too many details of services they use
- Services typically communicate via HTTPS using text-based payloads
 - Client makes GET, POST, PUT, or DELETE request
 - Body of the request is formatted as JSON or XML
 - Results returned as JSON, XML, or HTML
- Services should add functionality without breaking existing clients
 - Add, but don't remove, items from responses

REST

- Representational State Transfer
 - An architectural style
 - Scalable up to WWW and down to microservices
- Protocol independent
 - HTTP is most common
 - SMTP could be used
 - Others possible
- Service endpoints supporting REST are called RESTful
- Client and Server communicate with Request – Response processing

Features of a RESTful Service

- URIs (or endpoints) identify resources
 - Responses return an immutable representation of the resource information
- REST applications provide consistent, uniform interfaces
 - Stateless
 - Representation can have links to additional resources
- Caching of immutable representations is appropriate

Resources and Representations

- Resource is an abstract notion of information
- Representation is a copy of the resource information
 - Representations can be single items or a collection of items



This is Noir, he's a Schnoodle

This is Bree, she's a Mutt

Representation Formats

- JSON
- XML
- HTML
- YAML
- Something else ...

```
{"pets": [  
  {"name": "Noir", "breed": "Schnoodle"},  
  {"name": "Bree", "breed": "Mutt"}  
]}
```

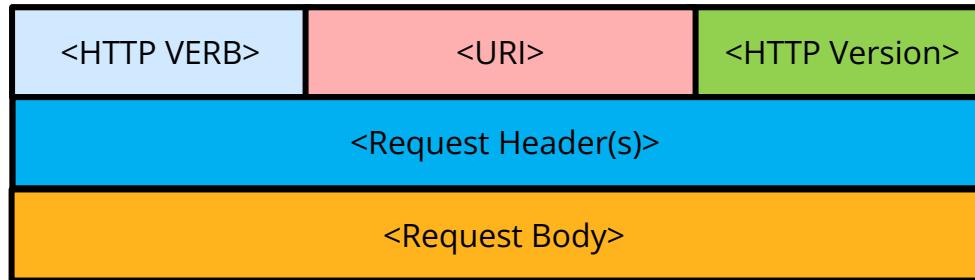
```
<pets>  
  <pet>  
    <name>Noir</name>  
    <breed>Schnoodle</breed>  
  </pet>  
  <pet>  
    <name>Bree</name>  
    <breed>Mutt</breed>  
  </pet>  
</pets>
```

```
<ul>  
  <li>  
    <h1>Noir</h1>  
    <div>Schnoodle</div>  
  </li>  
  <li>  
    <h1>Bree</h1>  
    <div>Mutt</div>  
  </li>  
</ul>
```

```
name, breed  
Noir, Schnoodle  
Bree, Mutt
```

```
pets:  
- name: Noir  
  breed: Schnoodle  
- name: Bree  
  breed: Mutt
```

HTTP Request



- VERB: method: GET, PUT, POST, DELETE, OPTIONS
- URI: Uniform Resource Identifier (endpoint)
- Request Header: metadata about the message
 - Preferred Representation formats (e.g., JSON, XML, etc.)
- Request Body: (Optional) Request state
 - Representation (JSON, XML) of resource

HTTP Request Examples

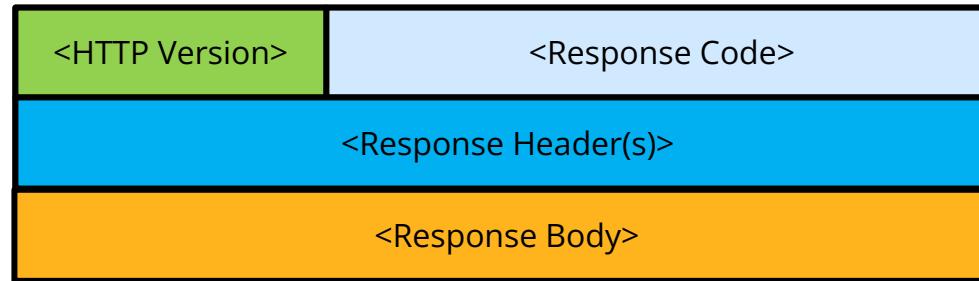
- GET

```
GET / HTTP/1.1  
Host: pets.drehnstrom.com
```

- POST

```
POST /add HTTP/1.1  
Host: pets.drehnstrom.com  
Content-Type: json  
Content-Length: 35  
  
{ "name": "Noir", "breed": "Schnoodle"}
```

HTTP Response



- Response Code: 3-digit HTTP Status code
 - 200 codes for success
 - 400 codes for client errors
 - 500 codes for server errors
 - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- Response Body: Contains Resource Representation

Recommendations for URI

```
URI = scheme "://" host "/" path [ "?" query ] [ "#" fragment ]
```

<https://www.roidtc.com/api/v1/pets>

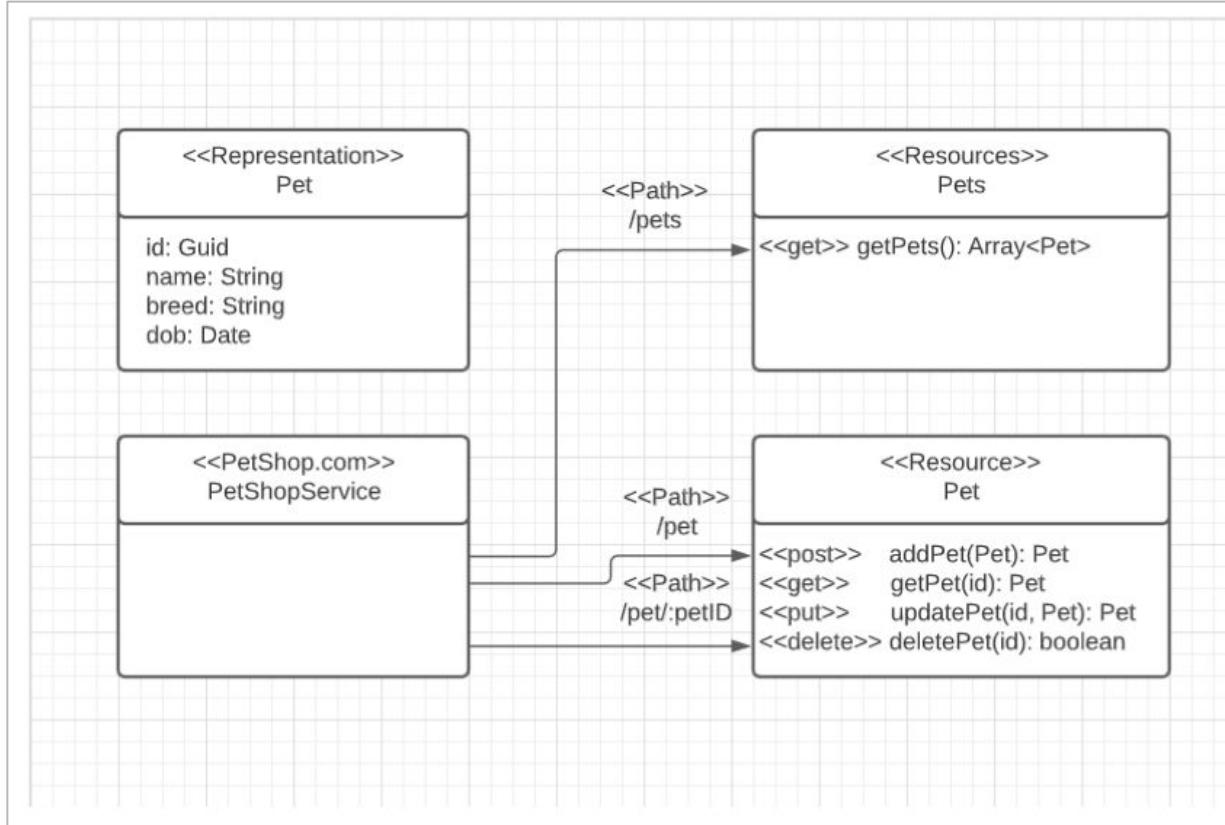
- URIs should not be used to indicate that a CRUD function is performed
 - Should uniquely identify resources and not any action upon them
- Include version information
- Use lowercase letters
- Don't use file extensions
- Strive for consistent naming

CRUD = create, read, update, and delete

HTTP Request Methods

- HTTP request methods (verbs) should be used to indicate which CRUD function is performed
 - **GET** is used to retrieve data
 - **POST** is used to add data
 - **PUT** is used to add data or alter/update existing data
 - *Put should be idempotent which means if the request is made once or multiple times, the effects on the data is exactly the same*
 - **DELETE** is used to remove data

Diagramming REST Services



OpenAPI

- Standard interface description format for REST APIs
 - Language agnostic
 - Open-source
 - Allows tools and humans to understand how to use a service without needing its source code
 - See:
[https://github.com/OAI/OpenAPI Specification](https://github.com/OAI/OpenAPI-Specification)

```
1  openapi: "3.0.0"
2
3  info:
4    version: 1.0.0
5    title: Swagger Petstore
6    license:
7      name: MIT
8
9  servers:
10   - url: http://petstore.swagger.io/v1
11
12 paths:
13   /pets:
14     get:
15       summary: List all pets
16       operationId: listPets
17       tags:
18         - pets
```

gRPC

- REST is great for communicating between services across the internet
 - Between browsers and web servers
 - Can be slow when communicating between microservices
- gRPC is a lightweight protocol for fast, binary communication between services or devices
 - Developed at Google
 - Supports many languages
 - Easy to implement
- See: <https://www.grpc.io/>

Activity: Programming Microservices

- Begin programming your case study microservices
 - Each microservice should be in its own code base
- A starting point will be provided
 - Written in Node.js
 - Broken into two services
 - External (Web frontend)
 - Internal (Backend)
- After you get the case study running:
 - As a group, come up with specific ways the case study application conforms to the twelve factors
 - There is a slide in your Google Slides document to record your results

Chapter Summary

In this chapter, you have:

- Designed systems using a microservice style architecture
- Followed Twelve-Factor App best practices
- Understood microservice communication protocols

Quiz

What is *not* an advantage of microservices over monolithic applications?

- A. Reduced risk when deploying new versions
- B. Services scale independently to optimize use of infrastructure
- C. Easier to innovate and add new features
- D. Can use different languages and frameworks for different services
- E. All of the above are advantages

Quiz

What is *not* considered a Twelve-Factor App best practice?

- A. Use a version control system like Git
- B. Keep secrets, connection strings, endpoints, etc. in source code for easier maintenance and deployments
- C. Strictly separate build, release, and run stages
- D. Keep development, staging, and production as similar as possible
- E. All of the above are advantages

Quiz

REST services most commonly use which protocol?

- A. FTP
- B. SOAP
- C. HTTP
- D. SMTP



ROITRAINING
MAXIMIZE YOUR TRAINING INVESTMENT™

Application Lifecycle Management

Chapter Objectives

In this chapter, you will:

- Manage application dependencies
- Maintain source code using Git
- Manage versions of code using branches and tags

Chapter Concepts

Package Management

Source Control

Version Control

Managing Application Dependencies

- Application prerequisites and dependencies should be declared in your code, but managed using a separate tool
 - Prerequisites should be downloaded automatically at build time
- Need to control versions over time
 - A new version of a dependency cannot break your code
 - You should upgrade your dependencies as you work on new versions
- Every modern language has one or more tools for managing dependencies
 - Maven or Gradle for Java
 - Pip for Python
 - NPM for Node.js
 - NuGet for .NET
 - Etc.

Maven

- Build automation tool for Java with plugins for other languages as well
 - Describes how the software is built
 - Manages project dependencies and makes sure they are up to date
 - <https://maven.apache.org/>
- Uses XML in pom.xml file to describe how the project is built
- Example pom.xml file

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Pip

- Python installer program
- Simple commands for install and uninstalling packages
 - `pip install requirement`
- Typically use an external file, requirements.txt, to declare dependencies
 - Can also declare dependency versions
 - `pip install requirements.txt`
- Example requirements.txt file:

```
requirements.txt x  
1 Flask==1.0.2  
2 pytest  
3
```

NPM

- Node.js applications use NPM for package management
 - List dependencies on package.json file
 - To install dependencies, run:
`npm install`
- Example package.json file:

```
package.json x
1  {
2    "name": "express-app",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "body-parser": "~1.18.2",
10     "cookie-parser": "~1.4.3",
11     "cors": "^2.8.4",
12     "debug": "~2.6.9",
13     "express": "~4.15.5",
14     "hbs": "~4.0.1",
15     "morgan": "~1.9.0",
16     "request": "^2.83.0"
17   }
18 }
```

Case Study Dependencies

- The case study app is written in Node.js
 - Currently is using NPM
 - Running `npm install` installs all dependencies defined in `package.json`
 - You already did this
- If you install a new dependency, npm can update your `package.json` file for you
 - For example, the following command will install the Google Firestore package and add it to the `package.json`:
`npm install @google-cloud/firestore`
 - The following will install the AWS SDK module:
`npm install aws-sdk`

Chapter Concepts

Package Management

Source Control

Version Control

Version Control System Types

- Centralized source control with file locks
- Centralized source with version merging
- Distributed source control

Centralized Source Control with File Locks

- One copy of the source code is maintained at the server
- Only one developer can edit a file at a time
 - Developers check out files for edit
 - Must check in files before others can edit
- Seems simple but is inefficient
 - Files are often left locked
 - Developers are left waiting for others to complete their work
- Visual SourceSafe is an example

Centralized Source with Version Merging

- One central repository, but all developers have a copy of the source
 - Anyone can change any file at any time
 - Developers need to synchronize changes before checking in code
 - File changes are merged when there are conflicts
- Subversion is an example

Distributed Source Control

- There can be many repositories
 - Each developer has their own repository
 - A central repository on some server becomes the origin for the others
- Developers check in code to their repository without affecting the origin
 - When code is ready, it can be pushed to the origin
 - Repositories must be synchronized prior to pushing to the origin
 - Conflicts must be resolved
- Git is an example of distributed source control

Git

- Free, open-source version control system
 - Created by Linus Torvalds for the Linux project in 2005
 - Has become the defacto standard for version control
 - <https://git-scm.com/>
- Advantages
 - Works on any platform and easy to install
 - Any folder can be a Git repository
 - Simple lightweight branching
- Web and cloud-based implementations exist
 - GitHub, Bitbucket, AWS CodeCommit, Google Cloud Source Repositories

Git Command Summary

Command	Description
git config --global user.name "Jeff Bezos" git config --global user.email jeff@amazon.com	Configure username and email. Required for commits.
git init	Initialize a local folder as a Git repository. Used for your working folder.
git clone /path/to/repository	Copy a repository.
git add <filename> git add *	Add files to the repository.
git commit -m "Commit message"	Commit changes to the local repository with a commit message.
git commit -a -m "Commit message"	Commit and add.
git push origin main	Push local commits to the remote repository main branch.
git checkout -b <branchname>	Create a new branch and switch to it.
git checkout <branchname>	Switch to a different branch.
git push origin <branchname>	Push a branch to the remote repository.
git pull	Pull changes from the remote repository to your local repository.
git status	Summary of which files have changes that are staged for the next commit

Tutorial: Using Git

If you are unfamiliar with Git, do the following for homework:

- Make sure Git is installed on your machine:
 - See: <https://git-scm.com/downloads>
- Do this Git tutorial:
 - See: <https://git-scm.com/docs/gittutorial>

GitHub

- Extremely popular, web-based Git service
 - Free to create unlimited public and private repositories
 - Paid accounts offer additional features
 - <https://github.com/pricing>

The screenshot shows a GitHub repository page for the user 'drehnstrom' with the repository name 'converter'. The page includes a navigation bar with links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Pulse, Graphs, and Settings. It also shows Unwatch (1), Star (0), Fork (0) buttons. A note says 'No description, website, or topics provided.' Below this are buttons for New and Add topics. Key statistics are displayed: 9 commits, 1 branch, 0 releases, and 1 contributor. A dropdown menu shows 'Branch: master' and a 'New pull request' button. At the bottom, there's a 'Create new file' button and a 'Clone or download' button. The commit history lists the following changes:

File	Commit Message	Time
.idea	Initial Commit	a day ago
templates	Added Celsuis and Fahrenheit	25 minutes ago
Converter.py	Initial Commit	a day ago
application.py	Added Celsuis and Fahrenheit	25 minutes ago
buildspec.yml	Edited requirements.txt again	14 hours ago
converterTests.py	Initial Commit	a day ago
requirements.txt	Edited requirements.txt again	14 hours ago

Tutorial: Using GitHub

- If you want more experience using GitHub, do the following tutorial for homework:
 - <https://guides.github.com/activities/hello-world/>

Google Cloud Source Repositories

- Provided as a tool on Google Cloud
 - Private and secured via Identity and Access management service
 - Integrates with other Google Cloud DevOps services like Container Builder and Container Repository

Development		Source Code
Source Code		converter
Repositories		/
Tools and plugins	Name	Latest Commit
	.idea	Initial Commit
	templates	Added Celsius and Fahrenheit
	application.py	Added Celsius and Fahrenheit
	buildspec.yml	Edited requirements.txt again
	Converter.py	Initial Commit
	converterTests.py	Initial Commit
	requirements.txt	Edited requirements.txt again

Amazon CodeCommit

- Git service provided by AWS

The screenshot shows the Amazon CodeCommit web interface. On the left is a sidebar with navigation links: Dashboard, Code, Commits, Commit Visualizer, Triggers, and Settings. The main area is titled "Code: converter". It displays a list of files in the "master" branch:

- converter
- .idea
- templates
- application.py
- buildspec.yml
- Converter.py
- converterTests.py
- requirements.txt

At the top right of the main area are buttons for "Clone URL" and "Connect". A question mark icon is also present in the top right corner.

Tutorial: Version Control with Git on the Cloud

If you like, here are tutorials on using Google Cloud Source Repositories and AWS CodeCommit

- [Cloud Source Repositories: Qwik Start](#)
- [Working with AWS CodeCommit](#)

.gitignore

- Git can be configured to ignore certain files
 - Do not include them in the repositories
- What kind of files should we have Git ignore?
- A `.gitignore` file is used to specify which files to ignore
 - Put this file in your local repo folder

Activity: Using Source Control

- Leverage Git to manage the source code of your case study app
 - Use a `.gitignore` file to ignore the node dependencies
- You will use GitHub, but any hosted Git repository would work
 - Put each service in a separate repository

Chapter Concepts

Package Management

Source Control

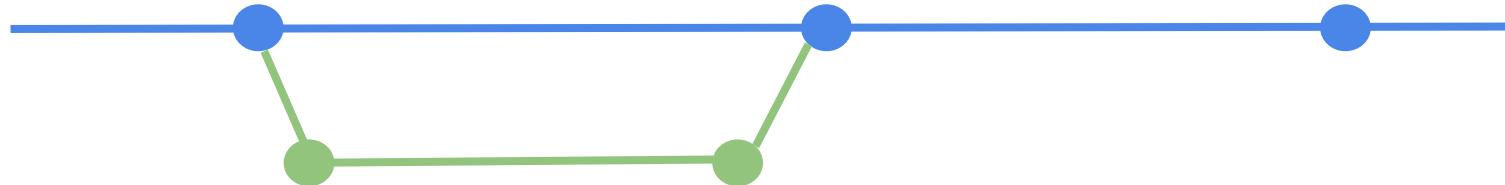
Version Control

Source Version Management

- There's not one "right" way to manage source versions over time
 - Use tags and branches to help organize code in a sensible way
- Every source code repository has a main branch
 - Was formerly known as the 'master' branch
 - Add branches to make code changes without affecting the main branch
 - Branches can be merged to incorporate changes into the main branch
 - Or branches can be discarded

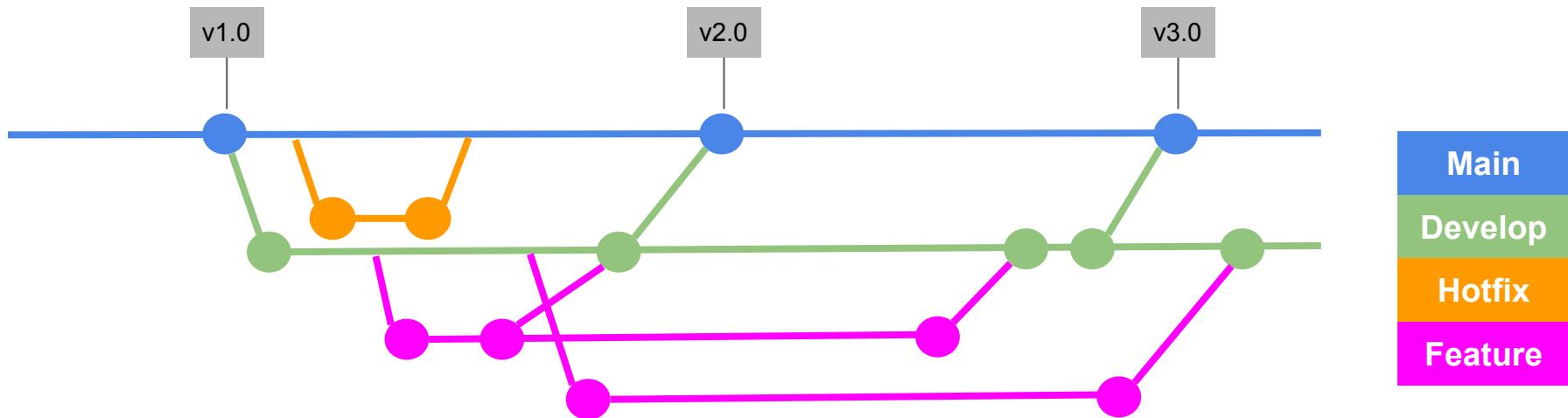
Branches

- Develop your code in a branch
 - Create a branch: `git branch my-branch`
 - Switch to a branch: `git checkout my-branch`
 - Commit changes: `git commit -a -m "Made Some Changes"`
- To merge back into the main branch:
 - Switch to main: `git checkout main`
 - Merge code changes: `git merge my-branch`



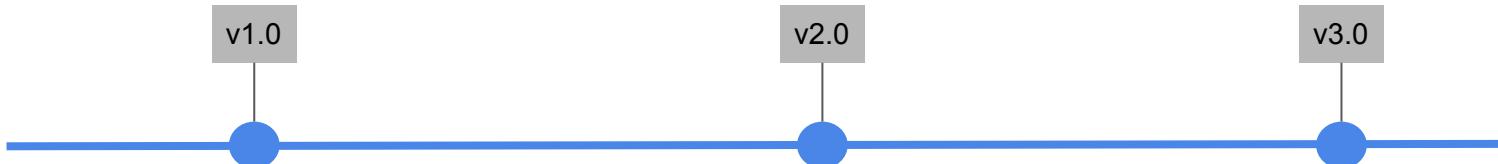
Branching and Merging Strategies

- The main branch contains the current production code
 - Add development branches for code you are working on
 - Add feature branches to development branches for each feature
 - When fixing a bug use a hotfix branch



Tags

- Use tags to track version history within a branch
 - `git tag -a v3.0 -m "My App version 3.0"`
- You can later check out code based on the tags
 - `git checkout v3.0`
- See: <https://git-scm.com/book/en/v2/Git-Basics-Tagging>



Activity: Branching and Collaboration

- Git branching
 - Create a branch
 - Make a change
 - Merge back into main
- Collaboration
 - Choose one of your team member's GitHub account to be the central repository
 - Make other team members collaborators on the repositories
 - Make changes and merge them to a shared repository

Chapter Summary

In this chapter, you have:

- Managed application dependencies
- Maintained source code using Git
- Managed versions of code using branches and tags

Quiz

You want to copy a GitHub repository onto your own computer. What is the easiest command to use?

- A. pull
- B. push
- C. merge
- D. clone

Quiz

You're using GitHub for source control and made some code changes. You ran git commit to commit your changes, but they don't show up in GitHub. What is likely the problem?

- A. You're probably not logged into GitHub correctly
- B. commit only saves changes to your local repository, you have to run git push as well to send changes to GitHub
- C. You need to run git merge not git commit
- D. You need to include the --synchronize parameter when running the command

Quiz

You're writing a program using Node.js and JavaScript. Which tool would you use to manage dependencies?

- A. NPM
- B. Pip
- C. Maven
- D. Gradle



Docker

Chapter Objectives

In this chapter, you will:

- Deploy microservice applications using containers
- Leverage Docker to build, run, and manage containers

Chapter Concepts

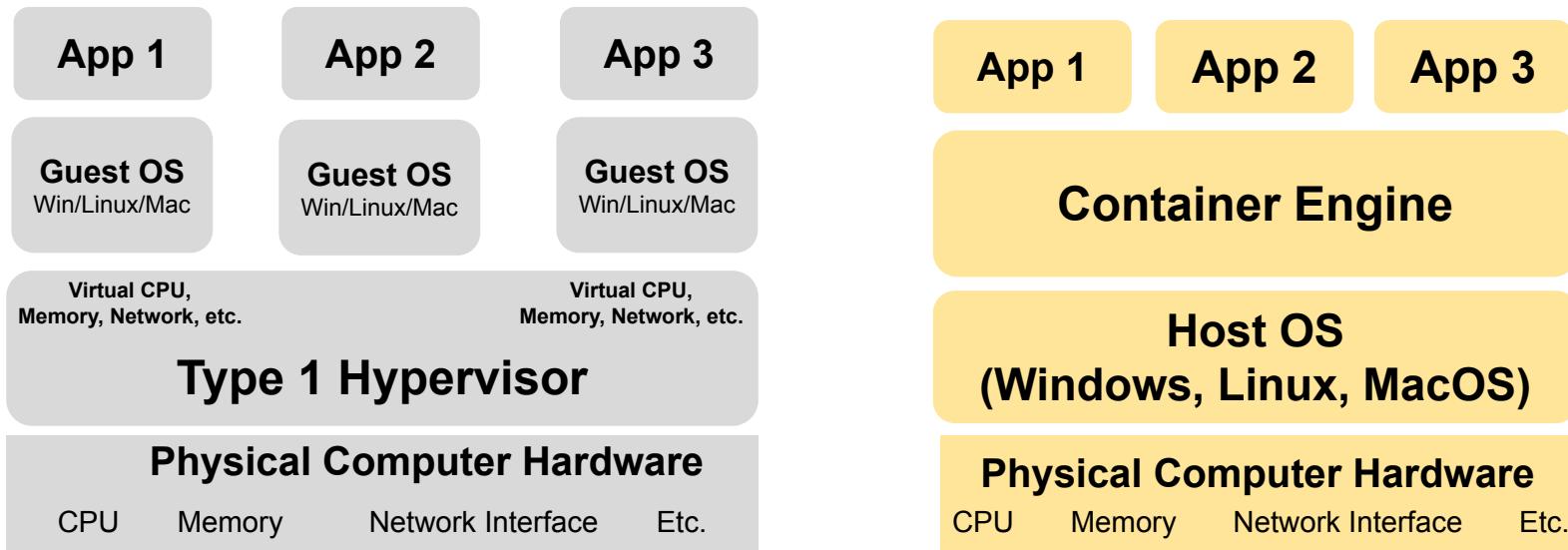
Understanding Docker

Using Docker

Deploying Docker Containers

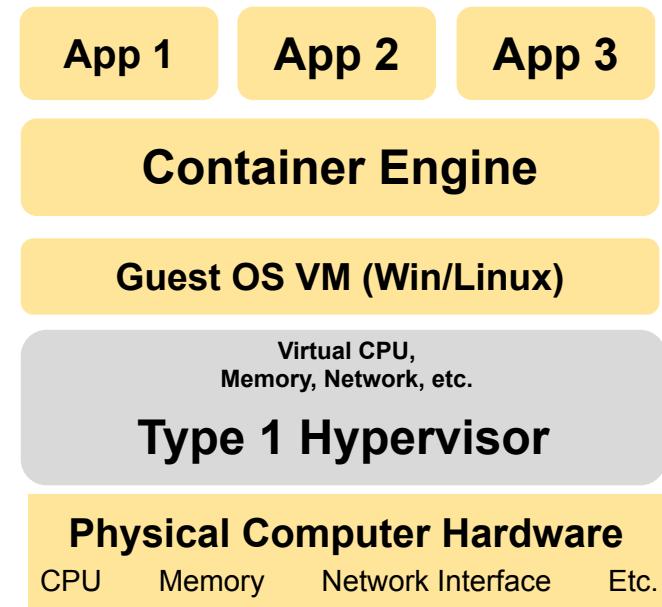
Containers

- Virtual machine-based virtualization is the ability to virtualize hardware
 - Resources of a single physical computer to be shared on multiple OSes
- Container-based virtualization is the ability to virtualize the OS kernel
 - Multiple isolated systems, called containers, access a single OS kernel



Containers (continued)

- Most containerized platforms today also use virtual machines
- Container engines run on top of virtualized servers
 - Leverages both hypervisor-based and container-based virtualization
 - Helps improve isolation and security



Docker

- Allows applications or microservices to be deployed to containers
 - Multiple containers can run on a single virtual machine
- Docker images are very lightweight, pre-configured virtual environments
 - Include the required software to run an application
 - Applications are inside the Docker image
- Docker images will run on any platform that has Docker installed
- Docker images allow applications to be easily moved
 - From developer to test to production environments
 - Between local and cloud-based data centers
 - Between different cloud providers

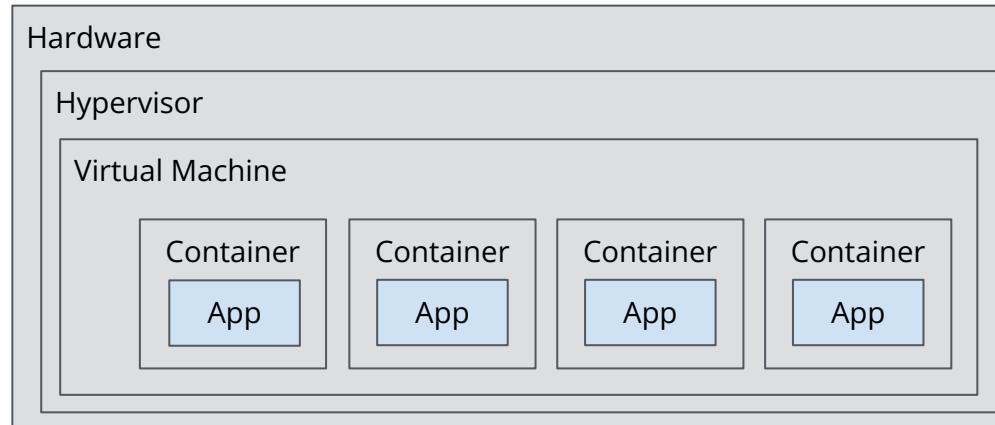
Images

- Images are deployment packages that are used to build containers
 - Containers are running instances of images
- Images are built in layers
 - Start with a base image
 - Add languages and frameworks used by your app
 - Copy in your code
 - Create environment variables
 - Specify how your application starts

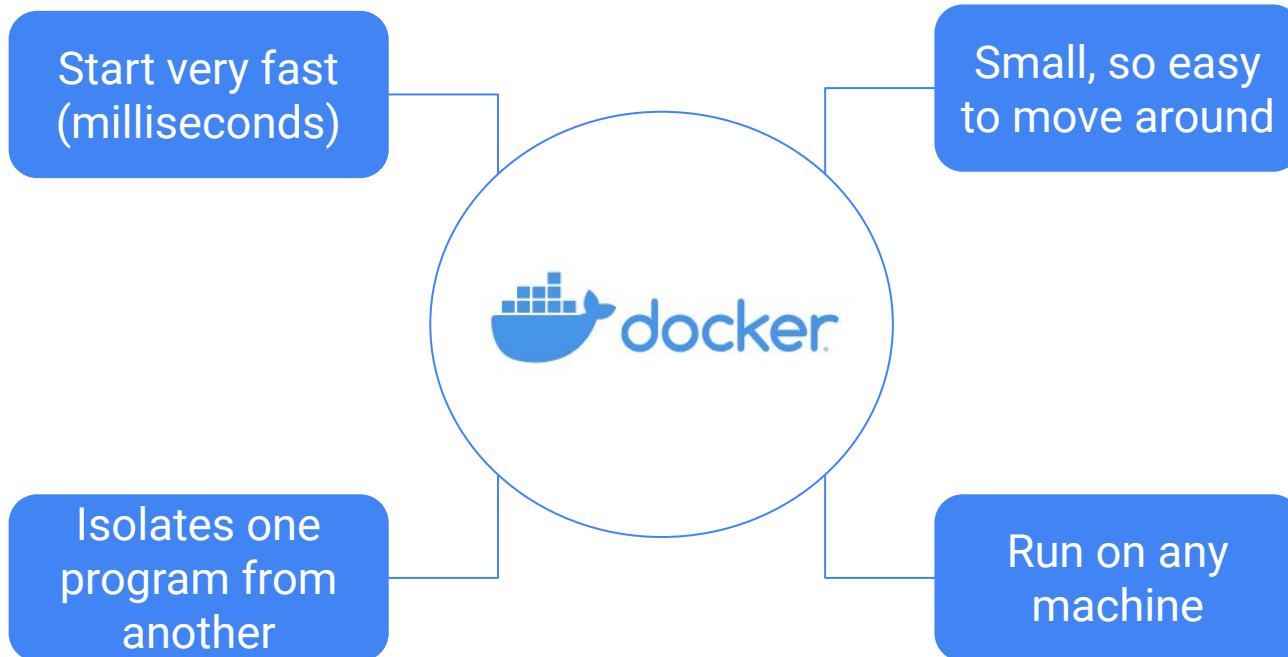


Containers

- Containers are running instances of images
- Containers do not include the operating system
 - The OS requires the container software be installed (Docker)



Advantages of Containers



Chapter Concepts

Understanding Docker

Using Docker

Deploying Docker Containers

Some Basic Docker Commands

Command	Description
docker build [OPTIONS] PATH URL - Example: docker build -t drehnstrom/converter-dar:latest .	Build a custom Docker container based on a Dockerfile . Run the command from the same folder as the Dockerfile
docker run [OPTIONS] IMAGE [COMMAND] [ARG...] Example: docker run -d -p 8080:8080 drehnstrom/converter-dar	Run a Docker image.
docker ps [OPTIONS]	List running docker images. Displays containers and their IDs.
docker stop [OPTIONS] CONTAINER [CONTAINER...] Example: docker stop <container-id-here>	Stop a running image.
docker login [OPTIONS] [SERVER]	Login to Docker Hub.
docker push [OPTIONS] NAME[:TAG] Example: docker push drehnstrom/converter-dar	Push a container to Docker Hub.
docker pull [OPTIONS] NAME[:TAG] Example: docker pull drehnstrom/converter-dar	Get a container from Docker Hub.

Creating Custom Docker Image

- To build a custom image, create a file call **Dockerfile**
- Steps
 1. Start with a base image from Docker Hub or another registry
 2. Identify yourself (*so you can upload your custom image later*)
 3. Install prerequisite software onto the base image
 4. Copy your application onto the image
 5. Configure your application
 6. Specify how to start your application
- Use `docker build` command to create the container
- Once the container is created use `docker run` command to start it

Example Dockerfile for a Node.js App

```
FROM launcher.gcr.io/google/nodejs
COPY . /app/
WORKDIR /app
RUN npm install
CMD ["npm", "start"]
```

Example Dockerfile for Python App

```
FROM python:3
WORKDIR /usr/src/app
COPY . .
RUN pip3 install -r requirements.txt
CMD [ "python3", "./main.py" ]
```

Example Dockerfile for a Java Spring Boot App

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG DEPENDENCY=target/dependency
COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app
ENTRYPOINT
["java","-cp","app:app/lib/*","hello.Application"]
```

Example Dockerfile for a .NET App

```
FROM microsoft/dotnet:latest
COPY ./ /app
WORKDIR /app
RUN ["dotnet", "restore"]
RUN ["dotnet", "build"]
EXPOSE 8080/tcp
ENTRYPOINT ["dotnet", "run", "--server.urls",
"http://0.0.0.0:8080"]
```

Example Dockerfile for an Nginx Website

```
FROM nginx
COPY ./ /usr/share/nginx/html
EXPOSE 80
```

Building Docker Images

- Use the Docker build command to create the image
 - -t parameter tags (*names*) the image (can include a version number)
 - Specify the path to the Dockerfile
- Tag is used later to specify which image you want to run
- Syntax:
 - `docker build -t your-docker-id/your-image:v0.1 .`

The `.dockerignore` File

- A `.dockerignore` file can be used to specify ignore rules and exceptions for the Docker COPY command
 - Place it in the root folder of the service
 - Similar to `.gitignore`
- A simple `.dockerignore` file for Node.js:

```
node_modules  
npm-debug.log
```

Example Build Command Output

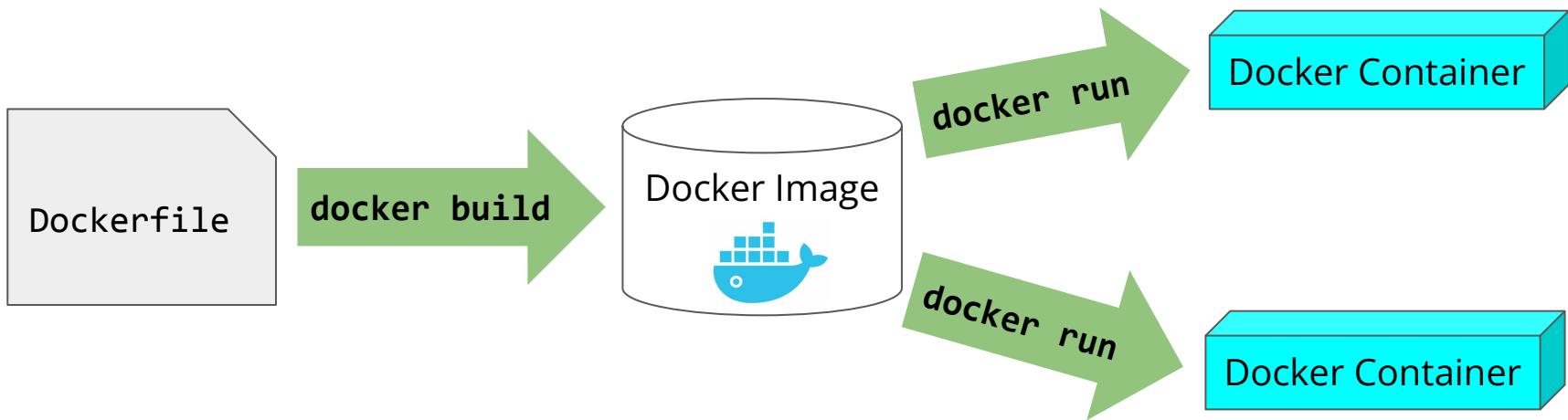
```
$ docker build -t drehnstrom/devops-demo:v0.1 .
Sending build context to Docker daemon 2.828MB
Step 1/7 : FROM python:3.7
--> 34a518642c76
Step 2/7 : WORKDIR /app
<< CODE OMITTED>>
Step 6/7 : ENV PORT=8080
--> Using cache
--> 7045daaafd44Step 7/7 : CMD exec gunicorn --bind :$PORT --workers 1 --threads 8
main:ap --> Using cache
--> 7c32a538632e
Successfully built 7c32a538632e
Successfully tagged drehnstrom/devops-demo:v0.1
```

Starting Containers

- Use the Docker run command to start a container based on an image
 - -p parameter specifies the port to listen on and the port to forward to
- Example:

```
$ docker run -p 8080:8080 drehnstrom/devops-demo:v0.1
[2019-07-02 12:07:13 +0000] [1] [INFO] Starting gunicorn 19.9.0[2019-07-02 12:07:13
+0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)[2019-07-02 12:07:13 +0000]
[1] [INFO] Using worker: threads[2019-07-02 12:07:13 +0000] [8] [INFO] Booting
worker with pid: 8
```

Docker Images and Containers



Listing Containers and Images

- To see your containers, use the Docker ps command
 - a parameter shows all containers, not just those that are running

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
7db7aed583f8      drehnstrom/devops-demo:v0.1   "/bin/sh -c 'exec gu..."
```

- To see your images, use the Docker Images command

```
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
drehnstrom/devops-demo   v0.1    7c32a538632e  23 minutes ago  946MB
python              3.7     34a518642c76  3 weeks ago   929MB
```

Deleting Containers and Images

- Use Docker `rm` command to remove containers
 - `docker rm <CONTAINER ID>`
- To stop all running containers:
 - `docker stop $(docker ps -a -q)`
- To remove all containers:
 - `docker rm $(docker ps -a -q)`
- Use `docker rmi` command to remove images
 - `docker rmi <IMAGE ID>`

Tutorial: Getting Started with Docker

- To learn more about Docker, do the following tutorial:
 - <https://docs.docker.com/get-started/>

Activity: Building Docker Containers

Containerize your case study project:

- Build Docker images for the Internal and External services
- Run your Docker images locally

Chapter Concepts

Understanding Docker

Using Docker

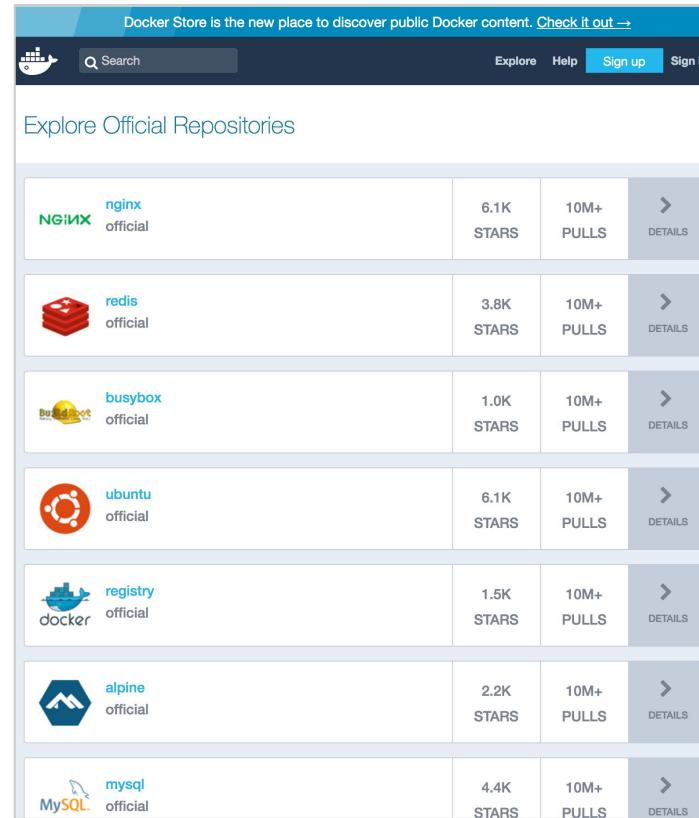
Deploying Docker Containers

Docker Registries

- Registries are centralized locations where Docker images can be stored
- Public registries are available to everyone
 - Base images for different environments are often stored publicly
 - Open-source applications might be stored in public registries
- Private registries are secured and managed by some organization
 - Control access to your proprietary software
- Registries are easy to create
- Access registries over the internet or your private network

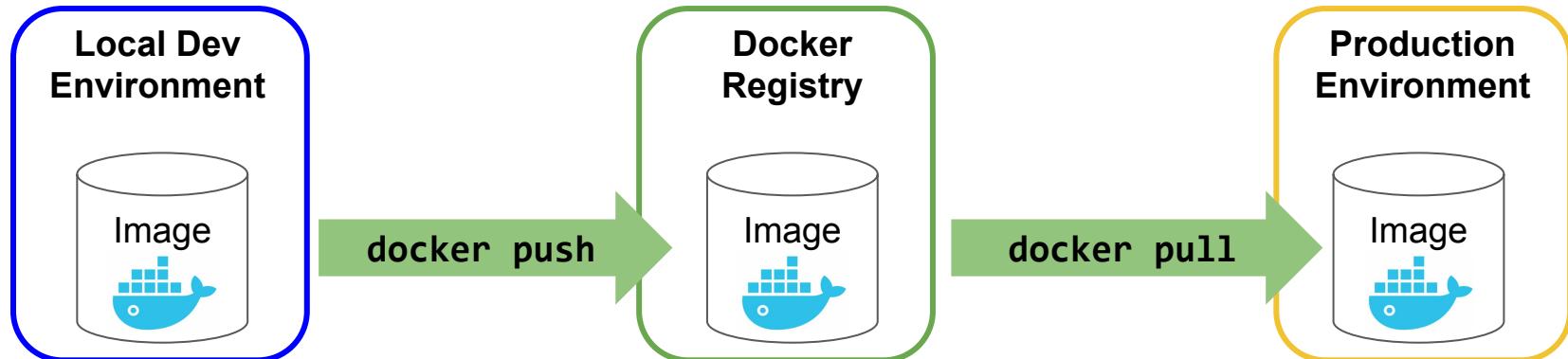
Docker Hub

- Official registry of Docker images
 - Can create both public and private Docker repositories
- Images for many operating systems and languages
 - Starting points for building your images
- Can upload custom images
 - When deployed onto systems, your custom images are downloaded from Docker Hub



Push and Pull

- Use the Docker push command to save an image to a repository
 - To save a container to Docker Hub:
`docker push your-docker-id/devops-demo:v0.1`
- Use the pull command to get an image from a repository
 - `docker pull your-docker-id/devops-demo:v0.1`



Running Your Own Container Registry

- Any server running Docker can act as a container registry
 - Just start the registry service and push images to it
- Allows complete control over storage of your Docker containers
- Example commands:

```
docker run -d -p 5000:5000 --name registry registry:2
```

```
docker pull ubuntu
docker tag ubuntu localhost:5000/myfirstimage
```

```
docker push localhost:5000/myfirstimage
docker pull localhost:5000/myfirstimage
```

AWS Elastic Container Registry (ECR)

ECR > Repositories

Repositories (1)			G	View push commands	Delete	Create repository
<input type="text"/> Find Repositories				<	1	>
Repository name	URI				Created at	▼
devops-demo	 213523735220.dkr.ecr.us-east-1.amazonaws.com/devops-demo				07/02/19, 10:26:48 AM	▼



```
$ docker push 213523735220.dkr.ecr.us-east-1.amazonaws.com/devops-demo:latest
```

Google Cloud Container Registry

- Docker registry provided by Google Cloud
 - Private and only available to those who are given access
- Container Builder service, creates containers and stores them
- To store images in Container Registry their names must be prefixed with `gcr.io/`
- Example command:

```
gcloud builds submit  
--tag gcr.io/gcp-project-id-here/container-name .
```

Qwiklabs: Container Registries

- AWS
 - [Introduction to Amazon Elastic Container Registry](#)
- Google Cloud
 - [Introduction to Docker](#)
 - [Container Registry: Qwik Start](#)

Activity: Container Registries

Use a Container Registry to store your Docker containers

- Push your containers to the registry
- Delete all local copies
- Run directly from the registry

Chapter Summary

In this chapter, you have:

- Deployed microservice applications using containers
- Leveraged Docker to build, run, and manage containers

Quiz

Running docker build results in what?

- A. A Docker container
- B. A Docker image
- C. A Docker repository
- D. A Docker instance

Quiz

If there was a Docker image stored in Docker Hub that you wanted to run on your computer, what Docker commands would you need?

- A. build and run
- B. push, build, and run
- C. pull, build, and run
- D. pull and run

Quiz

You want to create a Docker image for your application. What file do you need to create before running docker build?

- A. docker.yaml
- B. docker.json
- C. docker.conf
- D. Dockerfile



Kubernetes

Chapter Objectives

In this chapter, you will:

- Create Kubernetes clusters to host containers
- Use Kubernetes commands and configuration to deploy containers, services, autoscalers, and health checkers
- Secure Kubernetes applications
- Install applications with Helm

Chapter Concepts

Kubernetes Clusters

Kubernetes

Kubernetes Security

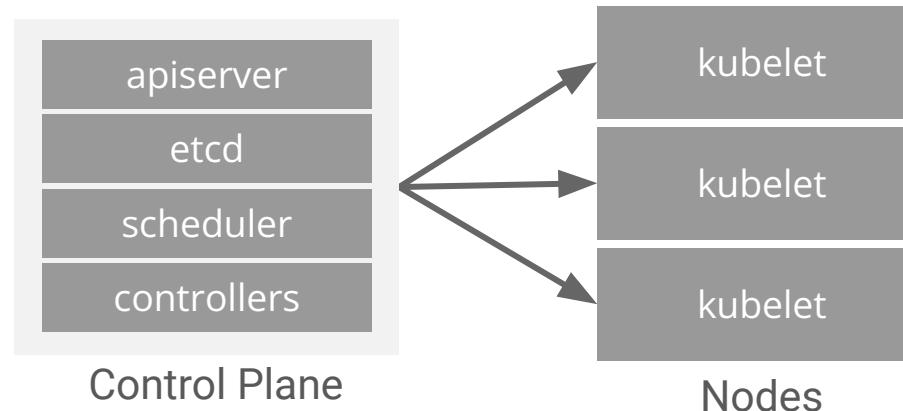
Helm

Kubernetes

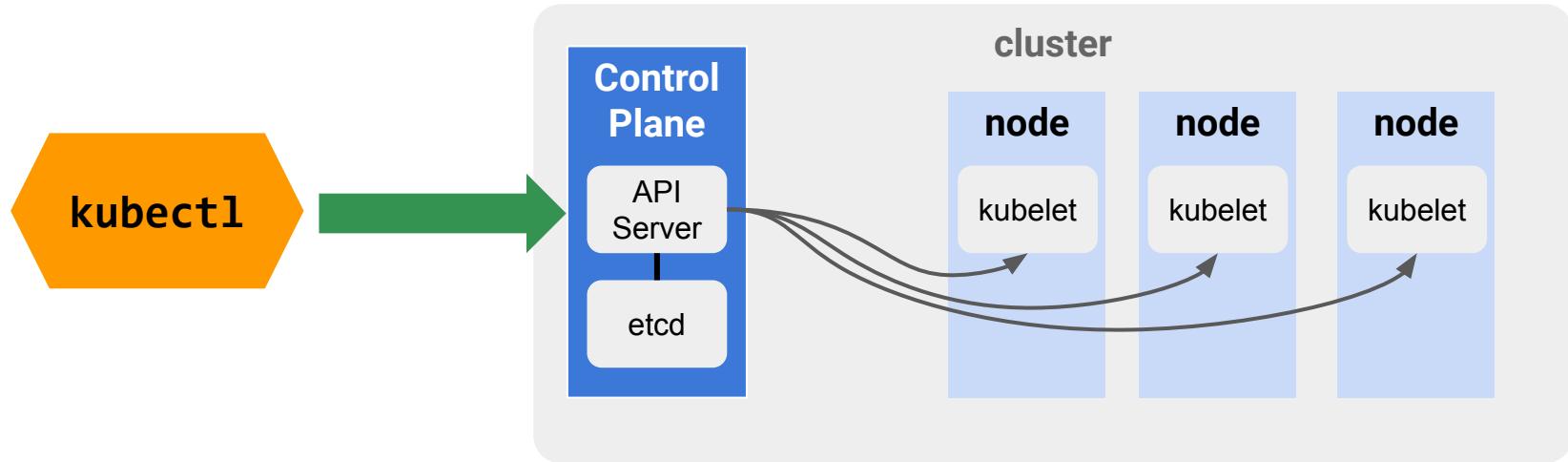
- Open-source container orchestration system
 - Extremely popular and active open-source project
- Originally developed by Google to run Google's data centers
 - Designed to operate at Google scale
 - Proven and tested running Google's applications
- Wide support
 - Used on Google Cloud when using Google Kubernetes Engine (GKE)
 - Supported by Microsoft in Azure Kubernetes Service (AKS)
 - Amazon Elastic Kubernetes Service (EKS)
 - Red Hat OpenShift
- Pivotal Cloud Foundry supports Kubernetes

Kubernetes Architecture

- Kubernetes applications require a cluster of machines to run on
 - One or more machines are designated as the control plane
 - Multiple machines are added to the cluster as nodes
- The control plane ensures applications are running on the nodes
- The nodes provide the computing and storage required by the applications



Kubernetes Clusters



Creating Kubernetes Clusters

Steps (*this is vastly over-simplified!*)

1. Buy some computers and install Linux on them

2. Install Kubernetes:

```
$ apt-get install -y kubeadm=##.# kubelet=##.# kubectl=##.#
```

3. Configure the control plane:

```
$ kubeadm init --kubernetes-version ##.# --pod-network-cidr  
192.168.0.0/16 ...
```

4. Grow the cluster by adding nodes:

```
kubeadm join --token ... --discovery-token-ca-cert-hash ...
```

5. Hire a couple IT guys to administrate the cluster

- Or, use a managed service to automate cluster creation

Amazon Elastic Kubernetes Service (EKS)

The screenshot shows the AWS EKS Clusters console. At the top, there's a navigation bar with 'EKS > Clusters'. Below it, a header says 'Clusters (1)'. On the right, there are three buttons: a grey 'Edit' button with a pencil icon, a grey 'Delete' button, and an orange 'Create cluster' button. A search bar with a magnifying glass icon and the placeholder 'Find clusters by name' is positioned below the header. To the right of the search bar are navigation arrows and a page number '1'. The main table lists one cluster:

Cluster name	Status
prod	ACTIVE

A modal window is open over the table, containing a terminal-style command:

```
$ eksctl create cluster \
--name prod \
--version 1.13 \
--nodegroup-name standard-workers \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami auto
```

Tutorial: Getting Started with Amazon EKS

- <https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html>

The screenshot shows the AWS Documentation website for Amazon EKS. The left sidebar has a navigation tree under 'Getting Started with Amazon EKS'. The main content area is titled 'Getting Started with eksctl' and describes the guide's purpose: to help install resources for managing Kubernetes clusters on Amazon EKS using eksctl. It includes sections for prerequisites, installing the AWS CLI, and a command-line example.

What Is Amazon EKS?

- Getting Started with Amazon EKS
 - Getting Started with eksctl
 - Getting Started with the Console
- Clusters
- Worker Nodes
- Storage Classes
- Load Balancing
- Networking
- Managing Cluster Authentication
- eksctl
- Service Limits
- Pod Security Policy
- Guest Book
- Metrics Server
- Prometheus Metrics

AWS Documentation » Amazon EKS » User Guide » Getting Started with Amazon EKS » Getting Started with eksctl

Getting Started with eksctl

This getting started guide helps you to install all of the required resources to get started with Amazon EKS using eksctl, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of this tutorial, you will have a running Amazon EKS cluster with worker nodes, and the kubectl command line utility will be configured to use your new cluster.

Prerequisites

This section helps you to install and configure the binaries you need to create and manage an Amazon EKS cluster.

Install the Latest AWS CLI

To use kubectl with your Amazon EKS clusters, you must install a binary that can create the required client security token for cluster API server communication. The `aws eks get-token` command, available in version 1.16.156 or greater of the AWS CLI, supports client security token creation. To install or upgrade the AWS CLI, see [Installing the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

If you already have pip and a supported version of Python, you can install or upgrade the AWS CLI with the following command:

Google Kubernetes Engine (GKE)

- Google's service for providing Kubernetes clusters
 - Google automates cluster creation and maintenance
 - Clusters are implemented as a collection of Compute Engine VMs
- Very simple creation using the web console or command line

```
$ gcloud container clusters create my-cluster --zone=us-central1-a  
--project=my-project-id
```

'Standard cluster' template

Continuous integration, web serving, backends. Best choice for further customization or if you are not sure what to choose.

Name

hip-local-cluster

Location type



Zonal



Regional

Zone

us-central1-a

Master version

1.11.7-gke.4 (default)

Node pools

Node pools are separate instance groups running Kubernetes in a cluster. You may add node pools in different zones for higher availability, or add node pools of different type machines. To add a

Tutorial: Getting Started with GKE

- <https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app>

broker
GKE dashboards
Continuous integration and delivery
Kubernetes comic

Tutorials
All tutorials
Deploying applications
[Deploying a containerized web application](#)
Create a guestbook with Redis and PHP
Using Persistent Disks with WordPress and MySQL
Authenticating to Cloud Platform with service accounts
Best practices for building containers
Deploying a language-specific application

Kubernetes Engine Tutorials

Deploying a containerized web application

This tutorial shows you how to package a web application in a Docker container image, and run that container image on a Google Kubernetes Engine cluster as a load-balanced set of replicas that can scale to the needs of your users.

Objectives

To package and deploy your application on GKE, you must:

1. Package your app into a Docker image
2. Run the container locally on your machine (optional)
3. Upload the image to a registry
4. Create a container cluster
5. Deploy your app to the cluster

☆ ☆ ☆ ☆ ☆

[SEND FEEDBACK](#)

Contents
Objectives
Before you begin
Option A: Use Google Cloud Shell
Option B: Use command-line tools locally
Set defaults for the gcloud command-line tool
Step 1: Build the container image
Step 2: Upload the container image
Step 3: Run your container locally (optional)
Step 4: Create a container cluster
Step 5: Deploy your ..

Azure Kubernetes Service (AKS)

Create Kubernetes cluster

Basics Scale Authentication Networking Monitoring Tags Review + create

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline. [Learn more about Azure Kubernetes Service](#)

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription [?](#) Pay-As-You-Go
└─ * Resource group [?](#) (New) aks-res-group
[Create new](#)

CLUSTER DETAILS

* Kubernetes cluster name [?](#) my-cluster
* Region [?](#) (US) East US
* Kubernetes version [?](#) 1.12.8 (default)
* DNS name prefix [?](#) my-cluster-dns

PRIMARY NODE POOL

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. You will not be able to change the node size after cluster creation, but you will be able to change the number of nodes in your cluster after creation. If you would like additional node pools, you will need to enable the "X" feature on the "Scale" tab which will allow you to add more node pools after creating the cluster. [Learn more about node pools in Azure Kubernetes Service](#)

* Node size [?](#) Standard DS2 v2
2 vcpus, 7 GiB memory
[Change size](#)

* Node count [?](#) 3

Tutorial: Getting Started with AKS

- <https://docs.microsoft.com/en-us/azure/aks/>

The screenshot shows the Microsoft Azure Documentation page for Azure Kubernetes Service (AKS). The left sidebar contains a navigation tree under the 'Azure / AKS' category, with sections for Overview, Quickstarts (Create an AKS Cluster, Develop applications), and 5-Minute Quickstarts (Use Draft, Use Java (VS Code & CLI), Use .NET Core (VS Code & CLI), Use .NET Core (Visual Studio 2017)). The main content area features a large heading 'Azure Kubernetes Service (AKS)' and a detailed description of what AKS does. Below the description is a section titled '5-Minute Quickstarts' with links to 'Azure CLI' and 'Azure Portal'.

Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline.

5-Minute Quickstarts

Learn how to deploy an AKS cluster:

Azure CLI Azure Portal

OpenShift

- Manages Kubernetes that runs in the cloud, on-premises or at the edge
 - Useful for private cloud deployments
 - Online, managed version is available

The screenshot shows the Red Hat OpenShift website. At the top is a black header bar with the Red Hat logo and navigation links: Products & solutions, Services & support, Resources, and Red Hat & open source. Below this is a white navigation bar with links for Red Hat OpenShift, Explore, Red Hat OpenShift products, Services and add-ons, and Help. The main content area has a black background and features the text "Red Hat OpenShift" in large white letters, followed by a description: "Red Hat® OpenShift® is an enterprise-ready Kubernetes container platform built for an open hybrid cloud strategy. It provides a consistent application platform to manage hybrid cloud, multicloud, and edge deployments."

Minikube

- Minikube is a free, open-source tool for running a Kubernetes cluster locally
 - Runs on Linux, Mac, and Windows
 - Useful for development and testing
- See: <https://github.com/kubernetes/minikube>
- Tutorial: <https://kubernetes.io/docs/tutorials/hello-minikube/>

Activity: Creating Kubernetes Clusters

You need a Kubernetes cluster to deploy your case study

- Using the cluster is exactly the same no matter where the cluster is running (Google, AWS, Azure, etc.)
- A cluster may be provided for you to use in the class
 - Your instructor will provide you with the details

Chapter Concepts

Kubernetes Clusters

Kubernetes

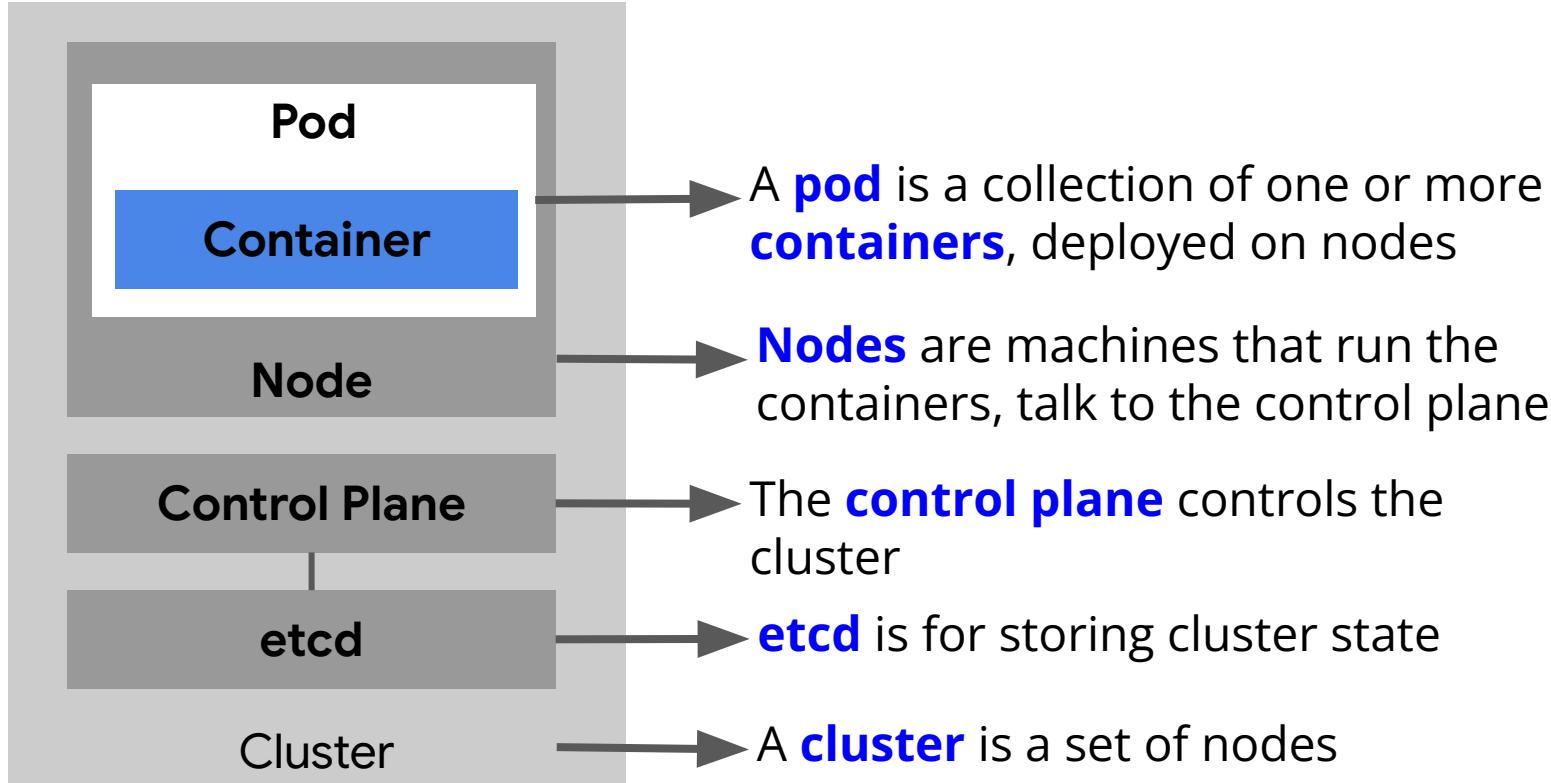
Kubernetes Security

Helm

Kubernetes Terms

- Pods are the smallest unit of deployment
 - Usually pods represent a single container
- Clusters are collections of machines that will run containers
 - Each machine is a node in the cluster
- Deployments are configurations that define service resources
- Replication controllers are used to create multiple instances of a pod
 - Guarantee pods are healthy and the right number exist
- Load balancers route requests to pods
- Autoscalers monitor load and turn pods on or off

Kubernetes Components



Kubernetes CLI

- Kubernetes is automated with a combination of CLI commands and configuration code
 - `kubectl` is the Kubernetes CLI
- Provided by the Cloud Native Computing Foundation
 - See: <https://kubernetes.io/>

kubectl Command	Description
kubectl get nodes	Show information about the cluster nodes
kubectl create -f config.yaml kubectl apply -f config.yaml	Create resources specified in the config.yaml
kubectl get pods kubectl describe pods	Show the running pods
kubectl get deployments kubectl describe deployments	Show the deployments
kubectl get hpa	Show horizontal pod autoscalers along with their min, max, and resource metrics
kubectl get services	Show running services along with their addresses and ports
kubectl delete [name] kubectl delete deployments/spaceinvaders	Delete specified resources
kubectl exec [options] [pod-name] kubectl exec -it spaceinvaders-55c88695bb-bcxb2 -- /bin/bash	Execute a command in the container (the example runs a bash shell)

Pods Can Be Defined in Configuration Files

- Pods are single instance
 - If a pod fails, it is not replaced automatically

```
apiVersion: v1
kind: Pod
metadata:
  name: devops-pod
spec:
  containers:
    - name: devops-demo
      image: drehnstrom/devops-demo:latest
      ports:
        - containerPort: 8080
```

Deployments Combine Pods with Replica Sets

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: devops-deployment
  labels:
    <Some code omitted to save space>
spec:
  replicas: 3
  selector:
    <Some code omitted to save space>
  template:
    <Some code omitted to save space>
  spec:
    containers:
      - name: devops-demo
        image: drehnstrom/devops-demo:latest
        ports:
          - containerPort: 8080
```

Kind of resource: Deployment

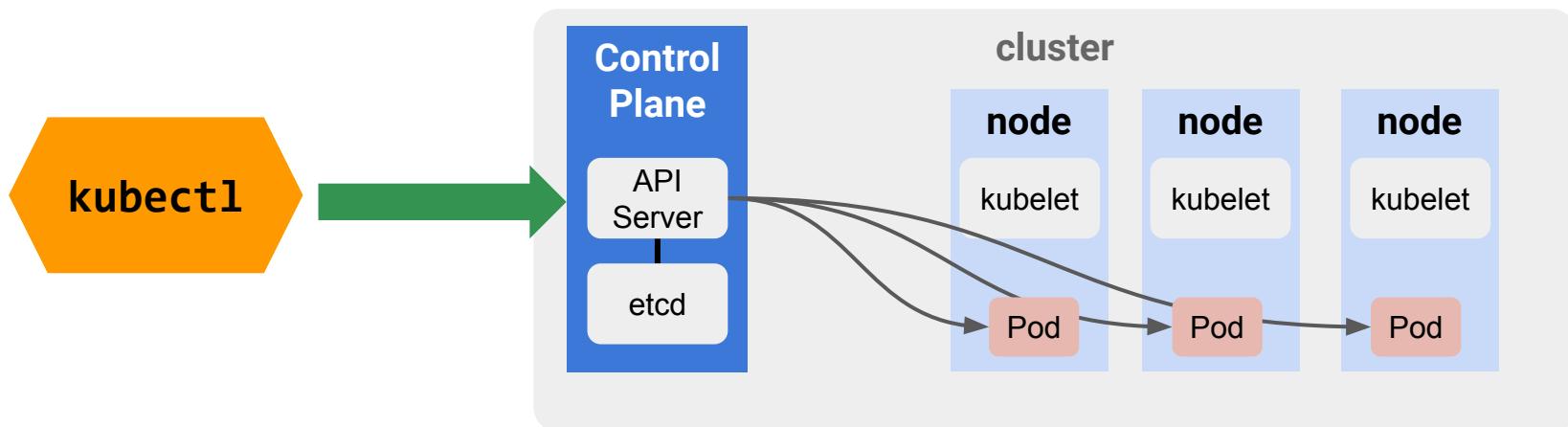
We want 3 replicas of the pod

This spec defines the pod. The same as the Pod configuration.

Kubernetes Deployments

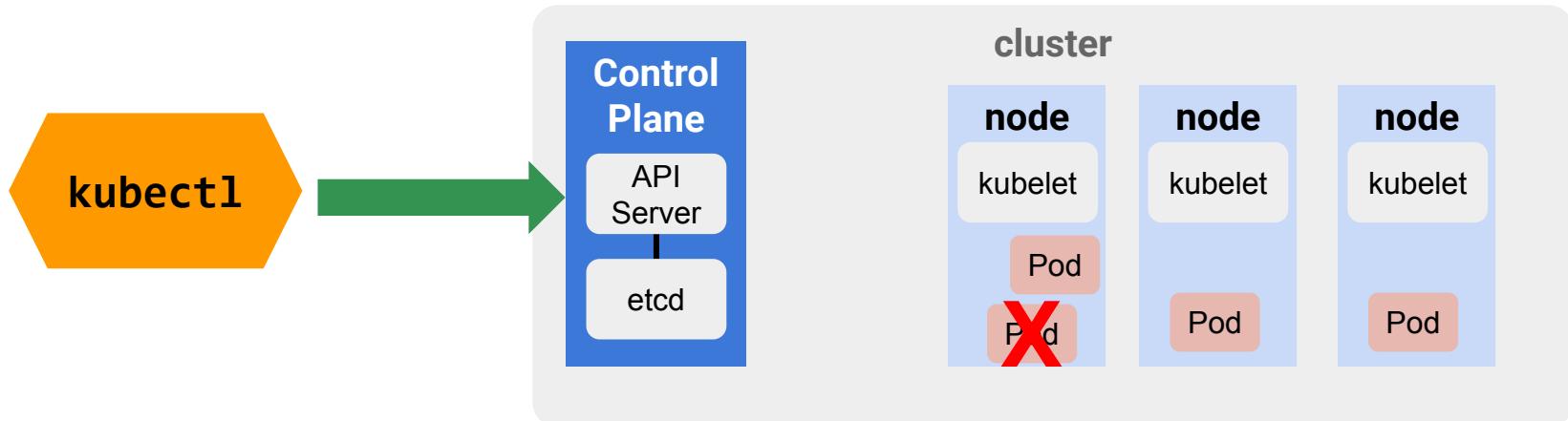
- Use kubectl to send the config file to the control plane
 - The control plane then decides how to deploy the pods

```
kubectl apply -f kubernetes-config.yaml
```



Kubernetes Replica Set

- A replica set ensures the correct number of pods is running
 - And replaces any pod that fails



Can Control the Resources Your Pods Require and Are Allowed to Consume

```
apiVersion: apps/v1
kind: Deployment

***CODE OMITTED FOR SPACE ***

spec:
  containers:
    - name: devops-demo
      image: drehnstrom/devops-demo:latest
      ports:
        - containerPort: 8080
  resources:
    requests:
      memory: "256Mi"
      cpu: "0.1"
    limits:
      memory: "512Mi"
      cpu: "0.5"
```

The minimum amount of resources required for each pod

The maximum amount of resources a pod is allowed to consume

Creating a Deployment from a Configuration File

- Deploy to a cluster based on a configuration file

```
kubectl apply -f kubernetes-config.yaml
```

- Show the running pods

```
kubectl get pods
```

- Show all the deployments

```
kubectl get deployments
```

- Show details of a deployment

```
kubectl describe deployments devops-deployment
```

Creating a Load Balancer via Configuration

```
apiVersion: v1
kind: Service
metadata:
  name: devops-loadbalancer
  labels:
    app: devops
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: devops
    tier: frontend
```

Creating a Load Balancer via Configuration (continued)

- Need a load balancer to route requests to the pods

```
kubectl apply -f kubernetes-config.yaml
```

- To get the load balancers IP address, use the following command:

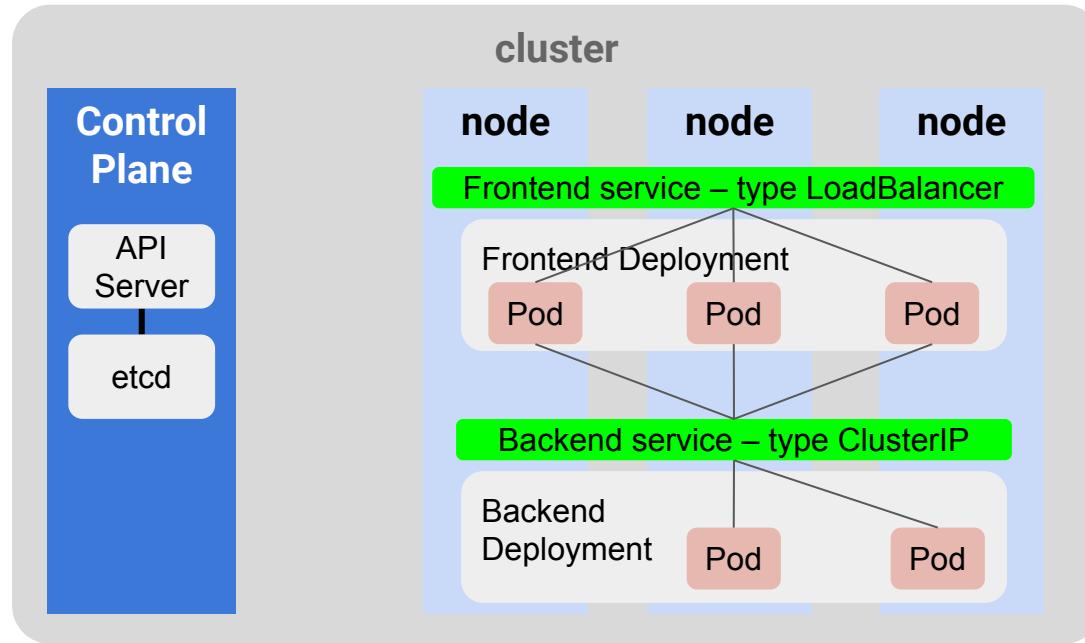
```
kubectl get services
```

Types of Services

ClusterIP	The default service type. Has only an internal IP address that is only accessible by other services running inside the cluster.
LoadBalancer	A service that provides an external IP address. In Google Cloud, this is implemented as a TCP load balancer. In AWS, this is implemented as an Elastic Load balancer. Not all Kubernetes deployments would support this type of service. Can be expensive if you have lots of services, which means lots of load balancers.
NodePort	Assigns a port between 30000 and 32767 to nodes in your cluster. When a node is accessed at that port, it routes to your service.

Types of Services (continued)

- A single application can have multiple services



Creating an Autoscaler via Configuration

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: devops-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: devops-deployment
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
  targetAverageUtilization: 60
```

Deleting Deployments and Resources

- Use the delete command to destroy anything previously created
 - Specifying a configuration file will delete everything created from it

```
kubectl delete -f kubernetes-config.yaml
```

- Can also delete resources individually when created at the command line

```
kubectl delete pod devops-pod  
kubectl delete deployment devops-deployment  
kubectl delete hpa devops-autoscaler  
kubectl delete services devops-loadbalancer
```

Health Checks

There are two types of Kubernetes health checks: Liveness and Readiness probes

- If a Liveness probe fails, the pod is deleted and re-created
- If a Readiness probe fails, requests are not sent to the pod until it is back up and running
 - But it is not removed and recreated

Liveness and Readiness Probe Configuration

```
apiVersion: apps/v1
kind: Deployment
*** CODE OMITTED ***
spec:
  containers:
    *** CODE OMITTED ***
    readinessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 30
      periodSeconds: 30
    livenessProbe:
      httpGet:
        path: /health
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 15
```

Added to the Containers section of the Deployment

If You Need to Debug a Pod, You Can Execute a Command Inside It

If something is wrong, run a bash shell in a pod

```
$ kubectl exec -it spaceinvaders-55c88695bb-bcxb2 -- /bin/bash
root@spaceinvaders-55c88695bb-bcxb2:/# curl http://localhost/
<!DOCTYPE html>

OUTPUT OMITTED

</body>
</html>
root@spaceinvaders-55c88695bb-bcxb2:/# exit
$
```

Activity: Deploying Applications with Kubernetes

Create Kubernetes configuration files to deploy your case study application on your Kubernetes cluster

- You will need to configure:
 - A deployment
 - A load balancer
 - An autoscaler (optional)

Optional Homework: Kubernetes Qwiklabs

- Below are some tutorials on using Kubernetes:
 - [Managing Deployments Using Kubernetes Engine](#)
 - [NGINX Ingress Controller on Google Kubernetes Engine](#)
 - [Running a MongoDB Database in Kubernetes with StatefulSets](#)
 - [Deploy a Web App on GKE with HTTPS Redirect using Lets Encrypt](#)

Chapter Concepts

Kubernetes Clusters

Kubernetes

Kubernetes Security

Helm

Securing a Kubernetes Cluster

- Use a minimal OS
- Use RBAC
- Use secrets for configuration

Use a Minimal OS

- Kubernetes nodes are VM instances
- The nodes should run a minimal OS to reduce any possible vulnerabilities
 - Container-Optimized OS (cos) from Google
 - Container-Optimized OS with containerd (cos_containerd)
 - Ubuntu
- Recommended for security to use the Container-Optimized OS
 - Optimized to enhance node security
 - cos_containerd is a variant of the cos image with containerd as the runtime
- Use a cloud provider's managed Kubernetes service

Role-Based Access Control (RBAC)

- RBAC is used to grant permissions to resources at the cluster or namespace level
 - RBAC allows you to define roles with rules containing a set of permissions
- When using RBAC, define roles with the desired permissions
 - The roles can then be bound to users
- RBAC permissions provide finer-grained control over access to resources within each cluster

RBAC Roles and ClusterRoles

- Permissions are defined within a Kubernetes Role or ClusterRole
 - A *Role* grants access to resources within a single namespace
 - A *ClusterRole* grants access to resources in the entire cluster
- A RoleBinding (or ClusterRoleBinding) grants the permissions in the Role (or ClusterRole) to a set of users
 - Contains a list of the users, and a reference to the Role (or ClusterRole) being granted to those users

Defining Roles and ClusterRoles

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: production
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Example Role in the “production” namespace that can be used to grant read access to pods

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: []
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Example ClusterRole to grant read access to secrets in any particular namespace, or across all namespaces, depending on how it's bound

Binding a Role or ClusterRole

```
kind: RoleBinding # must be RoleBinding or ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: production
subjects:
- kind: User
  name: steve@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role # must be Role or ClusterRole
  name: pod-reader # must match a Role or ClusterRole name to bind to
  apiGroup: rbac.authorization.k8s.io
```

This role binding assigns "steve@example.com" the pod-reader role in the "production" namespace

Kubernetes Secrets

- A secret is an object that contains a small amount of sensitive data
 - Such as a password, a token, or a key
- Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a docker image
 - A secret is only sent to a node if a pod on that node requires it
 - Not written to disk—stored in a tmpfs on the nodes
 - Deleted once the pod that depends on it is deleted
 - One pod does not have access to the secrets of another pod
 - Secrets are encrypted at the storage layer in etcd

Creating Kubernetes Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MlWYyZDF1MmU2N2Rm
```

Not encrypted, base64
encoded

```
$ kubectl get secret mysecret -o yaml
```

To get the secret

- For more info see: <https://kubernetes.io/docs/concepts/configuration/secret/>

Chapter Concepts

Kubernetes Clusters

Kubernetes

Kubernetes Security

Helm

Helm

- Helm is an open-source package manager for Kubernetes
 - Similar as apt-get and yum are package managers for Linux
- Organizes Kubernetes objects in packages called **charts**
 - A chart manages the deployment of complex applications
 - A chart is like a parameterized YAML template
 - When you install a Helm chart, Helm fills in the parameters you supply and deploys a release

Helm Repos

- A chart repository allows packaged charts to be stored and shared
- A community Helm chart repository is located at Artifact Hub
 - <https://artifacthub.io/>
- It is also possible to create and run your own chart repository
 - Creating your own is outside the scope of this course
 - For more information: https://helm.sh/docs/topics/chart_repository/
- To use a repo it must be added to the Helm client (initialized)

Using Helm

1. To use Helm, you first need a Kubernetes cluster with `kubectl` configured
2. Install the Helm client on the same system with `kubectl`
 - Helm provides binary releases for a variety of OSes
 - <https://github.com/helm/helm/releases>
3. Add a chart repository
 - Let's assume we want to install `redis` on a cluster
 - Go to <https://artifacthub.io/> and search for `redis`
 - Several charts will be located, click the desired one and it will provide the command to add the repository for that chart
 - For example:

```
helm repo add bitnami  
https://charts.bitnami.com/bitnami
```

Installing Applications with Helm

- To install a chart, you can run `helm install`
 - Provide the chart name

```
helm install my-redis bitnami/redis  
helm install my-apache bitnami/apache
```

- To see what is installed, run `helm list`

\$ helm list	NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART
	APP VERSION					
	my-apache	default	1	2021-09-26 15:56:46.782062447 +0000 UTC	deployed	apache-8.0.1
	2.4.46					
	my-redis	default	1	2021-09-26 15:56:58.910564104 +0000 UTC	deployed	redis-12.1.1
	6.0.9					

Activity: Using Helm to Install a Database

In this activity, you will:

- Use Helm to install a relational database to your cluster

Chapter Summary

In this chapter, you have:

- Created Kubernetes clusters to host containers
- Used Kubernetes commands and configuration to deploy containers, services, autoscalers, and health checkers
- Secured Kubernetes applications
- Installed applications with Helm

Quiz

Fill in the blanks: In Kubernetes, a _____ is a collection of one or more _____, deployed on _____ in a _____.

- A. container, images, VMs, computer
- B. pod, containers, nodes, cluster
- C. microservice, functions, platform, cloud
- D. service, containers, computers, swarm

Quiz

You're deploying a web application to a Kubernetes cluster. Which type of service would you use to provide a public IP address to the application?

- A. NodePort
- B. ClusterIP
- C. LoadBalancer
- D. DNSPort

Quiz

True or False: In a microservice application, each microservice should be deployed to its own Kubernetes cluster to maximize performance, scalability, and security and minimize the overall cost.

- A. True
- B. False



Cloud Data Services

Chapter Objectives

In this chapter, you will:

- Store object-based (files) data in cloud storage
- Deploy managed relational databases
- Leverage managed NoSQL data services
- Implement caching to improve performance
- Review some cloud-based data analytic tools

Chapter Concepts

Storing Binary Data

Relational Data Service

NoSQL Data Services

Caching

Data Warehousing and Analytics

Object-Based Storage

- Cloud providers offer managed services for object-based storage
 - Also called binary or Blob storage
- Extremely inexpensive
- Secure
- Designed for extremely high durability
 - 99.99999999%
- Objects are stored in buckets
 - No limit to the number of files stored in a bucket
 - No limit to the amount of data stored in a bucket
 - Maximum file size of a single object (file) is about 5 TB

Object-Based Storage (continued)

- Provides the following features:
 - Encryption at rest
 - Data fragmentation
 - Versioning and logging are easily enabled
 - Static website hosting
 - Deliver images, CSS, JavaScript, HTML, and other static files directly from a bucket
 - Cross-region storage
 - Store files across multiple geographic regions
 - Transfer acceleration
 - Faster data transfer for an added charge
 - Offline data imports
 - Can import large amounts of data with physical devices

Amazon Simple Storage Service (S3) Classes

- Standard
 - Objects copied to multiple zones within a region
 - 99.9% availability SLA
- Infrequent Access (IA)
 - Objects copied to multiple zones within a region
 - Lower storage costs but has a data retrieval cost
 - 99% availability SLA
 - Minimum 30-day storage duration
- One zone-IA
 - Stores data in a single AZ and costs 20% less than S3 Standard-IA
 - 99% availability SLA
 - Minimum 30-day storage duration

Amazon Simple Storage Service (S3) Classes (continued)

- Glacier/Glacier Deep Archive
 - Long-term archival
 - 99% availability SLA
 - Minimum 90-day/180-day storage duration
 - Cheapest storage costs with highest retrieval costs
 - Takes minutes/hours to retrieve data
 - “Faster” retrieval times cost more money
- Amazon S3 Intelligent-Tiering
 - Automatically moves data to the most cost-effective class
 - Without performance impact or operational overhead

Google Cloud Storage

- All data is encrypted at rest by default
 - Can optionally control encryption keys if required

Google Cloud Storage Classes

- Multi-Regional
 - Objects automatically copied to multiple regions in a geographic area
 - Up to 99.95% availability SLA
- Regional
 - Up to 99.9% availability SLA
- Nearline
 - Low storage cost with higher retrieval cost
 - Minimum 30 day storage duration
- Coldline
 - Low storage cost with higher retrieval cost
 - Minimum 90 day storage duration
- Archive
 - Minimum 365 day storage duration

All storage classes have millisecond retrieval time and use the same API

Azure Blob Storage Classes

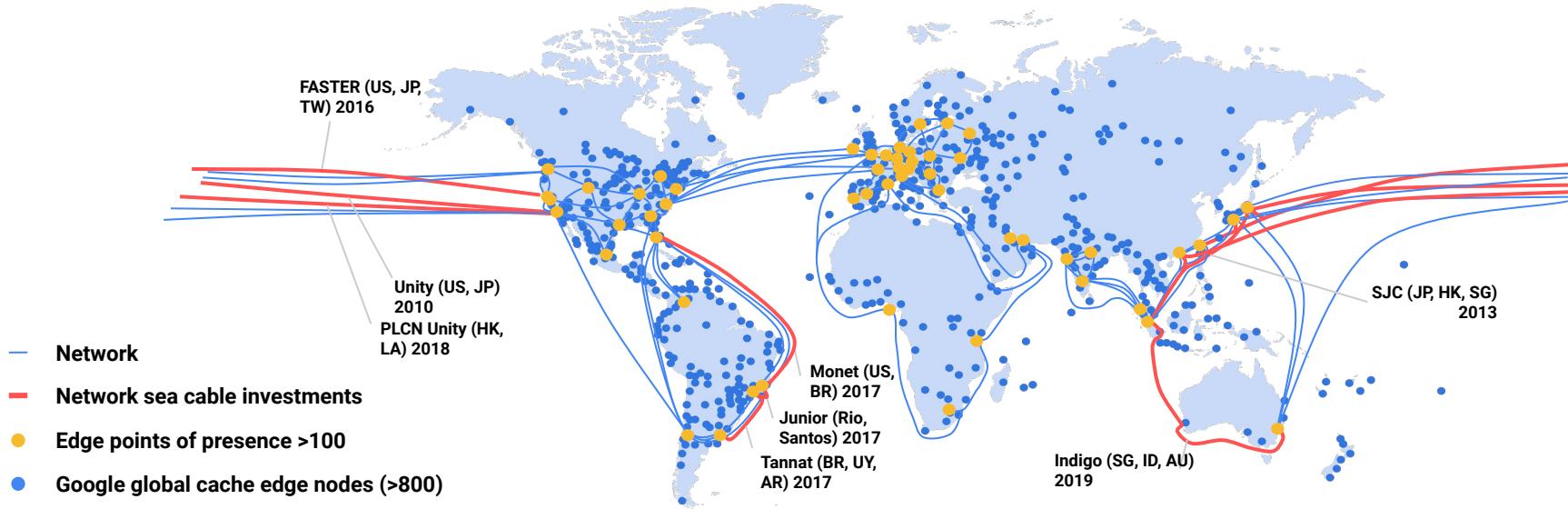
- Available in three replication classes
 - Locally Redundant Storage (LRS)
 - Replicated synchronously to three different storage nodes within the same region
 - Geo Redundant Storage (GRS)
 - Default option
 - Same as LRS plus asynchronous replication to another region
 - Read-Access Geo Redundant Storage (RA-GRS)
 - Read-only access to a storage account's data in the secondary region

Hosting Web Content

- Static web content can be delivered out of a Cloud Storage bucket
 - Useful for hosting content like images and videos

Caching Web Content

- Static web content can also be cached with a global CDN
 - Improves performance, reduces latency, and decreases egress cost
 - AWS, Google Cloud, and Azure all offer a CDN service



Qwiklabs: Cloud-Based Binary Storage

- Amazon S3
 - [Introduction to Amazon Simple Storage Service \(S3\)](#)
- Google Cloud Storage
 - [Cloud Storage: Qwik Start - CLI/SDK](#)

Chapter Concepts

Storing Binary Data

Relational Data Service

NoSQL Data Services

Caching

Data Warehousing and Analytics

Managed Relational Databases

- Cloud providers offer managed relational databases
 - No need to manually provision VMs, install or maintain database
 - High availability options also available



What are some advantages of relational database?

Managed Relational Databases (continued)

- Google
 - Cloud SQL offers MySQL, PostgreSQL, and MS SQL Server databases as a service
 - Cloud Spanner is a horizontally, global-scalable, and strongly consistent database
- AWS
 - Relational Database Service (RDS)
 - Supports multiple database systems including: MySQL, MariaDB, Oracle, SQL Server, PostgreSQL, Aurora
- Azure
 - SQL Server, MySQL, PostgreSQL databases as a service

Qwiklabs: Managed Relational Databases

- Amazon Relational Database Service (Amazon RDS)
 - Windows: [Introduction to Amazon Relational Database Service \(RDS\) \(Windows\)](#)
 - Linux: [Introduction to Amazon Relational Database Service \(RDS\) \(Linux\)](#)
- Google Cloud SQL
 - [Cloud SQL for MySQL: Qwik Start](#)
- Google Spanner
 - [Cloud Spanner: Qwik Start](#)

Chapter Concepts

Storing Binary Data

Relational Data Service

NoSQL Data Services

Caching

Data Warehousing and Analytics

NoSQL



What are some characteristics of NoSQL databases?

Types of NoSQL Database

- Key-value stores
 - Data is stored in key-value pairs
 - Examples include Redis, Dynamo, Oracle NoSQL database
- Document stores
 - Data is stored in some standard format like XML or JSON
 - MongoDB, CouchDB, and Domino are examples
- Wide-column stores
 - Key identifies a row in a table
 - Columns can be different within each row
 - Cassandra and HBase are examples

Managed NoSQL Databases

- Cloud providers offer managed non-relational (NoSQL) databases
- Google Cloud
 - Firestore/Datastore
 - Bigtable
- AWS
 - SimpleDB
 - DynamoDB
 - DocumentDB
- Azure
 - Cosmos DB

Google Bigtable Overview

- Wide-column NoSQL datastore similar to Cassandra or HBase
- Requires a cluster of machines be created
 - Cluster can be easily resized for scalability
- Each row in a Bigtable table must have a unique key
 - Only index available is the key
 - Very fast data retrieval as long as the key is searched
- Bigtable is optimized for extremely fast writes
- Use Bigtable when application accumulates huge amounts of data quickly
 - IoT applications, mobile apps, games, etc.
- See: <https://cloud.google.com/bigtable/>

Google Bigtable Features

- Massively scalable by adding nodes to Bigtable cluster
- High availability by creating clusters in multiple zones
- Supports open-source HBase API
- Very simple programming
- Ideal for applications with a huge volume and very fast writes
 - Internet of Things (IoT)
 - High-volume web or mobile analytics

Google Cloud Datastore

Things to know:

- Completely managed document store
 - No administration, no maintenance, nothing to provision or set up
- 1GB per month free tier
- Indexes created for every property by default
 - Secondary indexes and composite indexes are supported
- Supports ACID transactions
- Schemaless
- For pricing info see: <https://cloud.google.com/datastore/pricing>

Google Cloud Firestore

- Firestore is the new and improved version of Datastore
 - Reworking of Firebase Realtime Database
 - Two modes: Native and Datastore
- Native mode supports all Firebase features
 - Uses Firebase API
 - Not supported with older App Engine runtimes
- Datastore mode does not support all Firestore features like offline support for mobile devices and synchronization
 - Compatible with Datastore API
- Older Datastore database will be migrated to Firestore in Datastore mode
- See: <https://cloud.google.com/datastore/docs/>

AWS NoSQL Options

- Amazon DynamoDB
 - Supports both document and key-value store models
 - Very fast and scalable
- Amazon DocumentDB
 - Fully-managed MongoDB-compatible database service
- Amazon Keyspaces
 - Serverless Cassandra-compatible database

Qwiklabs: NoSQL Data Services

- AWS
 - [Introduction to Amazon DynamoDB](#)
- Google
 - [Datastore: Qwik Start](#)

Storage Options Review

-  Which storage option(s) would be the best choice for an application that stores large video files?

-  Which storage option(s) would be the best choice to store huge amounts of data streamed from IoT devices?

-  Which storage option(s) would be the best choice to quickly migrate an on-premises Oracle database to the cloud with the least effort?

Comparing Storage Options

Object/BLOB

NoSQL

SQL

Good for: Structured and unstructured binary or object data	Good for: Flat data, heavy read/write, events, analytical data	Good for: Web frameworks, existing applications
Use cases: Images, large media files, backups	Use cases: User profiles, IoT data, web applications, microservices	Use cases: User credentials, customer orders, product catalogs

Chapter Concepts

Storing Binary Data

Relational Data Service

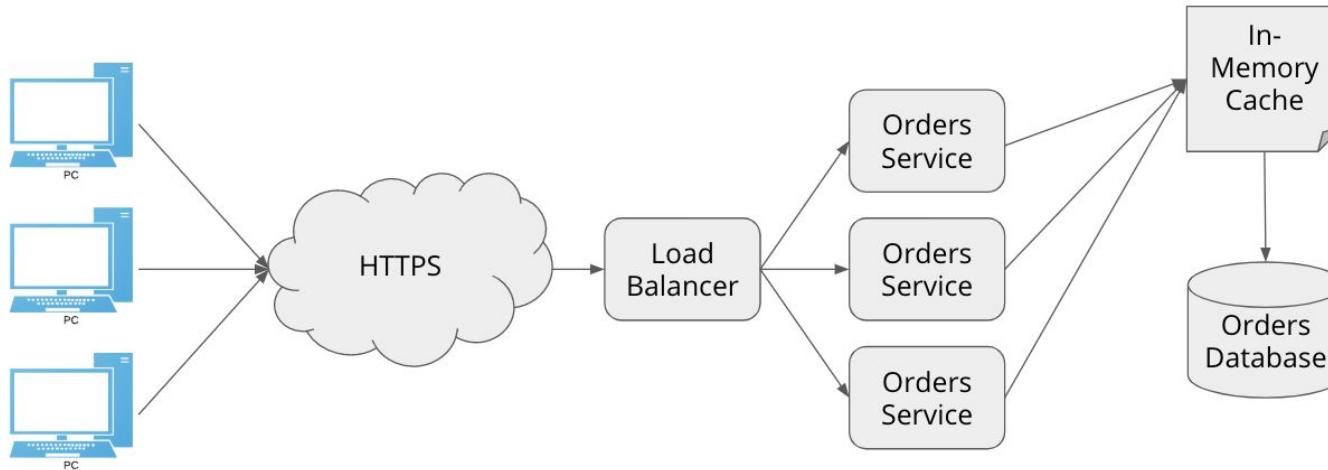
NoSQL Data Services

Caching

Data Warehousing and Analytics

Caching Database Content

- In the diagram below, the database is a potential bottleneck
 - Use a caching service like Redis
- Clients look for data in the cache for fast access
 - If the data is not cached, get it from the database



Caching Systems

- Redis
 - Open-source in-memory data structure store
 - Used as a database, cache, and message broker
- Memcached
 - Open-source distributed-memory object caching system
 - In-memory key-value store for small chunks of arbitrary data

Cloud-Based Managed Caching Services

- Amazon ElastiCache
 - Managed, Redis or Memcached-compatible in-memory data store
- Google Memorystore
 - Fully-managed in-memory data store service for Redis or Memcached

Activity: Using Data Services

- Configure the case study to use a database

Chapter Concepts

Storing Binary Data

Relational Data Service

NoSQL Data Services

Caching

Data Warehousing and Analytics

Data Warehousing and Analytics

- Data warehouses or data lakes combine data from various sources
 - Systems allow data to be analyzed quickly and stored inexpensively
 - On-premises solutions require hardware investment, specialized administration, and high-licensing costs
- Data analysts require simple and fast data analysis services

Amazon Redshift

- Petabyte-scale data warehousing solution
- Massively parallel processing
- Columnar storage reduces query I/O
- Compression reduces storage cost
- Need to configure resource allocation
 - Pay for what you allocate, not what you use

Amazon Athena

- Completely managed service for querying data stored in S3
 - Data is simply put into CSV, JSON, or Avro files and stored in S3 bucket
 - Specify table Schemas
 - Write standard ansi-sql queries to analyze the data
- Extremely inexpensive
 - Storage is just S3
 - Analysis cost is just \$5/TB of data processed
- <http://docs.aws.amazon.com/athena/latest/ug/what-is.html>

Google BigQuery

- Massively scalable, no-ops, inexpensive data warehousing
 - Import data from CSV, JSON, or Avro files
 - Storage cost is 2 cents/GB/Month for first 3 months, then 1 cent/GB/Month after that
- Extremely easy-to-use data analysis
 - Simply write SQL queries
 - \$5/TB processing cost
- <https://cloud.google.com/bigquery/docs/>

Chapter Summary

In this chapter, you have:

- Stored object-based (files) data in cloud storage
- Deployed managed relational databases
- Leveraged managed NoSQL data services
- Implemented caching to improve performance
- Reviewed some cloud-based data analytic tools

Quiz

**You want to store transactional data for customer orders and accounts.
Which type of storage service would be best?**

- A. Object storage
- B. Relational database
- C. NoSQL database
- D. Data warehouse

Quiz

You want to store static files such as JPEGs, PDFs, stylesheets, and JavaScript files for delivery via a web application. Which type of storage service would be best?

- A. Object storage
- B. Relational database
- C. NoSQL database
- D. Data warehouse

Quiz

You want to gather data from different sources such as databases, log files, web traffic data, and others for use in a data analytics and business intelligence project. Which type of storage service would be best?

- A. Object storage
- B. Relational database
- C. NoSQL database
- D. Data warehouse



Platform as a Service (PaaS)

Chapter Objectives

In this chapter, you will:

- Deploy an application into a managed Platform as a Service (PaaS)
- Leverage serverless platforms to deploy microservice logic

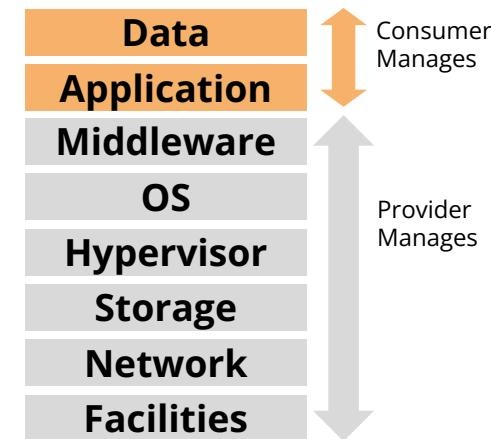
Chapter Concepts

Automated Platforms

Serverless Platforms

Platform as a Service (PaaS)

- Delivery of a hosted application environment as a service
 - Provides resources required to deploy and execute an application
 - Including auto-scaling, load balancing, health checks, etc.
- No need to buy or maintain resources required to execute the software
 - Simply deploy your application code into the environment
 - Enables organizations to concentrate on area of expertise
 - And not worry about managing IT infrastructure
 - Downside is code must conform to rules of the platform
 - Might require re-work for existing applications
- Many combinations of services by different PaaS providers



AWS Elastic Beanstalk

- Supports a number of different platforms
 - Select from a predefined environment
 - Or can use a custom Docker container
- Upload code changes into an environment
 - Code is automatically deployed
- Applications run on EC2 instances
- Automatically provisions autoscaling and load balancing

✓ Select a platform
Preconfigured
.NET (Windows/IIS)
Java
Node.js
PHP
Python
Ruby
Tomcat
Go
Packer
Preconfigured – Docker
GlassFish
Go
Python
Generic
Docker
Multi-container Docker

Google App Engine

- Google App Engine is available in two versions
 - Standard
 - Flexible
- Standard environment uses Google-specific containers for deployment
 - Can scale in 200ms and will scale down to zero instances when there are no users
 - Supports Java, Python, PHP, Node.js, Ruby, and Go
- Flexible environment uses Docker containers for deployment
 - Supports custom Docker images
 - Docker containers run on Compute Engine virtual machines

Azure App Service PaaS

- Azure App Service provides a PaaS
 - Web Apps, Web Apps for Containers, Mobile Apps, and API Apps
 - Supports Java, Node.js, PHP, Python, .NET, and Ruby
 - Automatically provisions auto scaling and load balancing
- Code can be deployed via:
 - Visual Studio (web deploy)
 - GitHub
- Provides staging slots for testing and rollback

Tutorials: Deploying to a PaaS

- Google Cloud App Engine
 - [Deploy Node.js Express Application in App Engine](#)
- AWS Elastic Beanstalk
 - [Getting started using Elastic Beanstalk - AWS Elastic Beanstalk](#)

Chapter Concepts

Automated Platforms

Serverless Platforms

Serverless/NoOps Computing

- Cloud native moves from an Infrastructure as a Service (IaaS) solution to a serverless/no-ops solution to deploy code
 - There are still servers, we just don't deploy or manage them
 - Nothing to provision
 - Nothing to "optimize"
 - Just use them and they work
- Can be very cost effective
 - Only pay when code actually runs
 - Most cloud providers offer perpetual free tiers

Serverless/NoOps Computing (continued)

- AWS Lambda, Google Cloud Functions, Azure Functions
 - Ability to execute code in response to an event
 - No need to deploy servers and install application
 - Only pay when the code runs
- Google Cloud Dataflow
 - Fully-managed service for transforming and processing data (ETL)
 - Can work in streaming (real-time) and batch (historical) modes
 - Work is performed on massively parallel managed clusters

AWS Lambda

- Supports functions written in virtually any language
 - Natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby
 - Custom AWS Lambda runtimes allows functions to be written in any programming language
- AWS Lambda imposes limits on functions
 - Must complete within 15 minutes
 - Maximum memory allocation of 3008 MB
 - 512 MB of /tmp directory storage
 - Max deployment package size of 50 MB (zipped) or 250 MB (unzipped)

Google Cloud Functions

- Supports functions written in:
 - Go, Java, Node.js, Python, Ruby, PHP, .Net Core
- Imposes limits on functions
 - Must complete within 9 minutes
 - Maximum memory allocation of 2048 MB
 - Max deployment package size of 100 MB (zipped) or 500 MB (unzipped)

Triggering Serverless Functions

- AWS Lambda and Google Cloud Functions can be triggered in several ways
 - An HTTP call
 - In response to a file added to a storage bucket
 - When a message is posted to a message queue
 - Etc.

Other Serverless Services

- Google Cloud Run
 - Managed platform for stateless containers
 - Can run as a fully managed, serverless environment
 - Or as a PaaS in your own GKE cluster
- AWS App Runner
 - Fully managed service to deploy from source code or container image

Optional Demo: Deploy a Microservice



Your instructor may choose to do one or more of the following:

- Deploy to Google App Engine
- Deploy a container in Google Cloud Run
- Use an AWS Lambda function or Google Cloud Function to implement a microservice that is called automatically when “something” happens

Tutorial: Leveraging Serverless Environments

- AWS Lambda
 - [Introduction to AWS Lambda](#)
- Google Cloud Functions
 - [Cloud Functions: Qwik Start - Console](#)
- Google Cloud Run
 - [Hello Cloud Run](#)

Chapter Summary

In this chapter, you have:

- Deployed an application into a managed PaaS
- Leveraged serverless platforms to deploy microservice logic

Quiz

Why might you choose a Platform as a Service offering such as App Engine or Elastic Beanstalk over Kubernetes?

- A. Complete control over the underlying operating system
- B. Easier and more automated
- C. More portable across cloud platforms
- D. All of the above

Quiz

AWS Lambda, Google Cloud Functions, and Azure Functions are examples of which type of environment?

- A. PaaS
- B. Serverless
- C. Containerized
- D. Unmanaged



DevOps Automation (CI/CD)

Chapter Objectives

In this chapter, you will:

- Automate builds and deployments using CI/CD pipelines
- Build a CI/CD pipeline with Jenkins

Manual Steps for Deployment

- So far, we have deployed everything manually
 - Nothing is automated
- What would you need to do if you had to update your application?
 - Pull current code from central source repo to local repo
 - Create new branch for feature
 - Make changes and commit to local repo
 - Run tests locally
 - Pull from central repo to ensure consistency, then push to central repo
 - Upload container to container registry
 - Update Kubernetes deployment files with new image tag (version)
 - Deploy to Kubernetes
 - Verify execution on Kubernetes

Chapter Concepts

Automated Builds

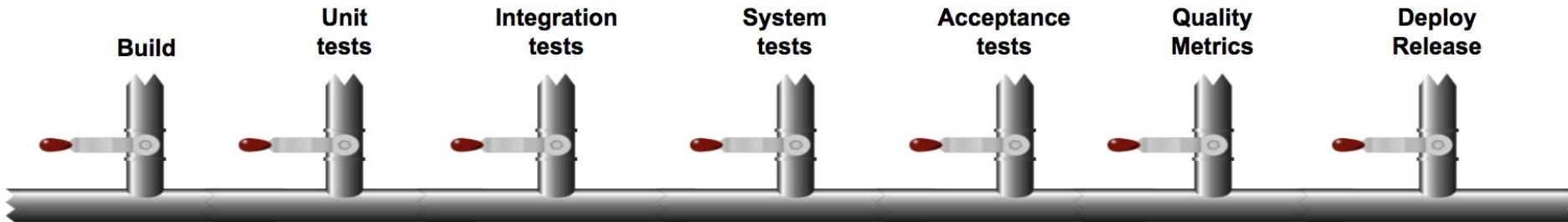
Jenkins

Continuous Integration

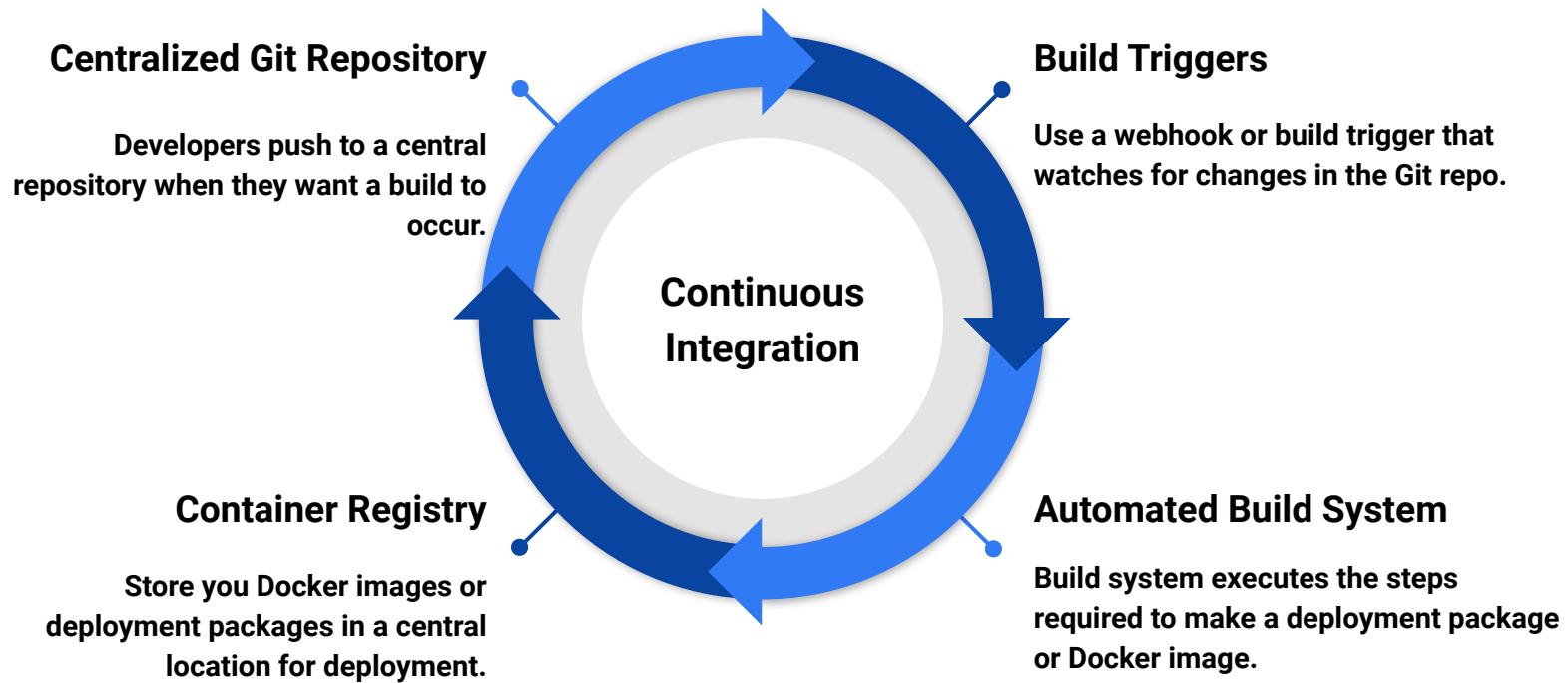
- Continuous integration is ultimately about automation
 - Tools used to make testing, quality, and deployment fast and simple
- Deploying new software or new versions should be no big deal
 - Deployments are done frequently to test environments
 - Deployment to production should be as simple as a configuration change
- Requirements for continuous integration
 - Source control
 - Infrastructure as code
 - Build automation
 - Test automation
 - Automated deployment
 - Visibility and reporting

Continuous Delivery

- Releases into production are automated based on quality metrics
 - Software is released whenever it meets the quality standards
- Small releases happen very frequently
 - Less risky than big bang releases for both customers and developers
- Continuous delivery pipeline is used to manage the deployment process
 - Steps are defined within the pipeline
 - Some quality metric is used to trigger the next step
 - The final step is deploying a new release into production



Continuous Integration Workflow



Automating Builds

- When a developer pushes their code to the main repository, then the current container is out of date
 - Rebuild it
 - Use the latest version of the container for testing
 - When testing is complete and the code is ready, deploy it

GitHub Webhooks

- GitHub uses webhooks to automate builds
 - Create some endpoint that responds to repository actions
 - When code is pushed, a message is sent to the endpoint

The screenshot shows the 'Webhooks / Add webhook' page on GitHub. On the left, there's a sidebar with options like Options, Collaborators, Branches, Webhooks (which is selected and highlighted in orange), Integrations & services, and Deploy keys. The main area has a heading 'Webhooks / Add webhook'. It explains that a POST request will be sent to the specified URL with event details. It also links to developer documentation. There are fields for 'Payload URL *' (containing 'https://www.mysite.com/build'), 'Content type' (set to 'application/x-www-form-urlencoded'), and a 'Secret' field. A note says 'By default, we verify SSL certificates when delivering payloads.' with a 'Disable SSL verification' button. At the bottom, there's a section for selecting events: 'Just the push event.' (radio button checked), 'Send me everything.', and 'Let me select individual events.'

AWS Code Automation

- AWS provides a number of tools for automating builds
 - AWS CodeCommit is their Git repository
 - AWS CodeBuild defines how a build should occur
 - AWS CodePipeline orchestrates a series of steps in a deployment

AWS CodeBuild

- AWS CodeBuild pulls source code from a repository, executes a set of build commands, and tests and writes results to some output location
- Builds are defined in the `buildspec.yml` file
 - Essentially a list of commands and a list of files to copy

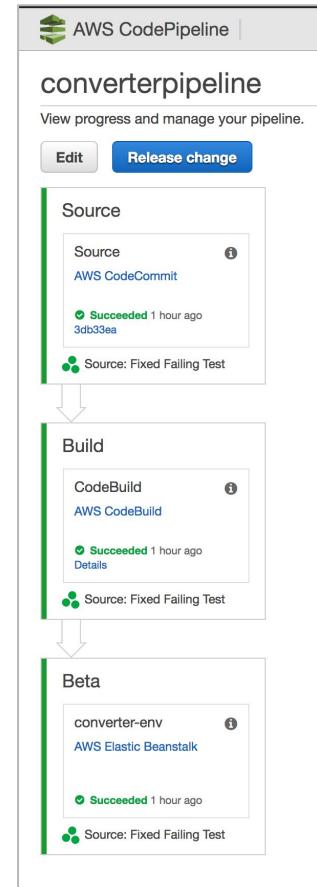
The screenshot shows the AWS CodeBuild console interface. On the left, there's a sidebar with 'AWS CodeBuild' at the top, followed by 'Build projects' (which is highlighted with an orange border) and 'Build history'. In the center, the title 'Build Project: converter' is displayed above a form. At the top of the form are three buttons: 'Back' (gray), 'Delete' (red), and 'Edit project' (blue). Below the buttons, the project details are listed:

Project name	converter
Description	None
Source provider	AWS CodeCommit
Repository	https://git-codecommit.us-east-1.amazonaws.com/v1/repos/converter
Artifacts upload location	converter-project

At the bottom of the form, there's a link labeled '▶ Project details'.

AWS CodePipeline

- Defines a deployment as a series of steps
 - Watches a repository waiting for a change
 - When the repository changes, perform a build
 - If the build succeeds, do a deployment
- Supports AWS CodeCommit or GitHub repositories
- Can deploy to various deployment platforms including AWS Elastic Beanstalk, AWS CloudFormation, and others



Tutorial: AWS CodeBuild

Do the following if you want to learn about AWS CodeBuild:

- [Getting started with AWS CodeBuild using the console - AWS CodeBuild](#)

Google Cloud Code Automation

- Google Cloud provides a number of tools for automating builds
 - Cloud Source Repositories is a hosted Git repository
 - Cloud Build defines how a build should occur
 - Container Registry for storing Docker containers

Cloud Source Repositories

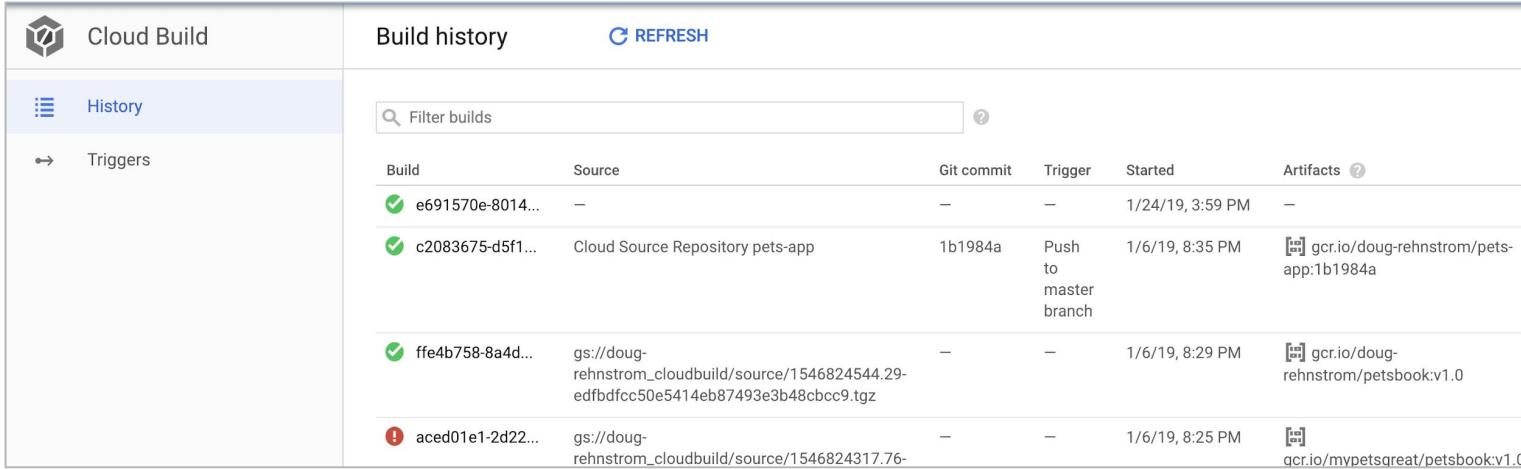
The screenshot shows the Google Cloud Source Repositories interface. At the top, there's a navigation bar with a logo, the text "Cloud Source Repositories", a search bar labeled "Everything" and "Search for code or files", a help icon, and a "Cloud Console" link. On the far right of the bar is a user profile icon with a letter "D". Below the bar, the main area is titled "My Source". It features a "Starred (6)" section containing a list of repositories, each with a star icon, a copy icon, the repository name, and the owner's name. The repositories listed are: converter-cloud-function, devops-converter, image-index, pets-app, space-invaders, and streaming-pipeline, all owned by doug-rehnstrom.

- ★ ⌂ converter-cloud-function doug-rehnstrom
- ★ ⌂ devops-converter doug-rehnstrom
- ★ ⌂ image-index doug-rehnstrom
- ★ ⌂ pets-app doug-rehnstrom
- ★ ⌂ space-invaders doug-rehnstrom
- ★ ⌂ streaming-pipeline doug-rehnstrom

Google Cloud Build

- Google hosted Docker build service
 - Alternative to using Docker build command
- To submit a build enter the following from the folder with the Dockerfile

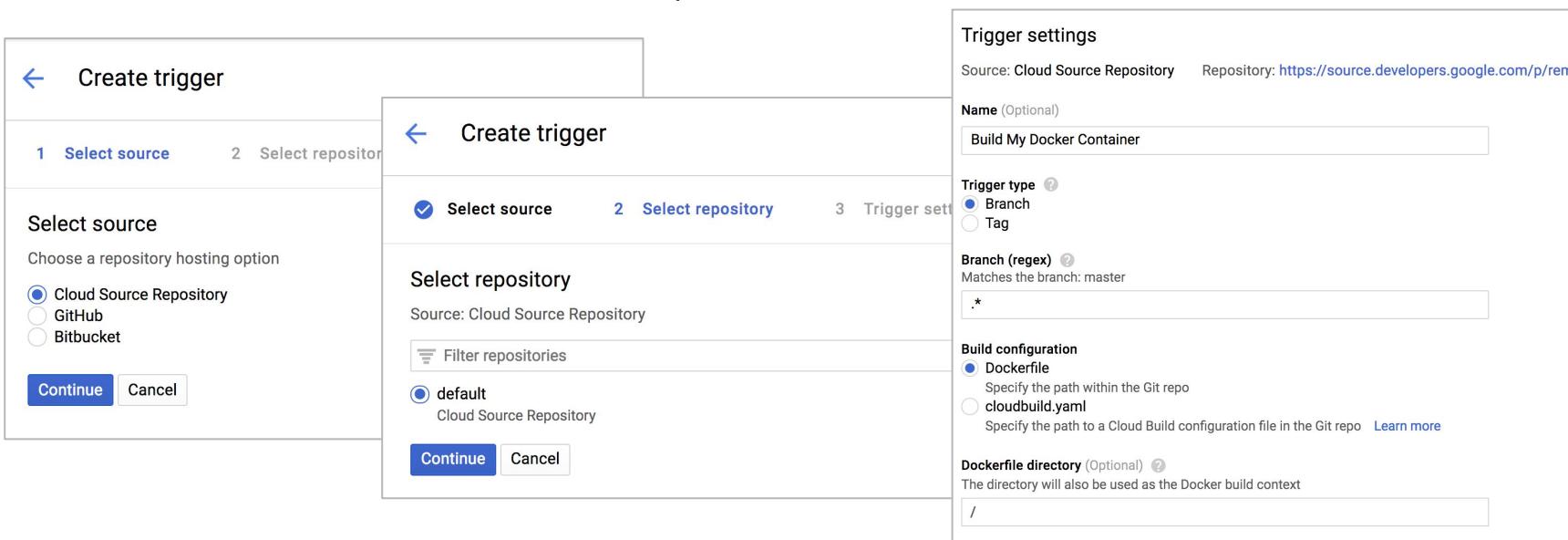
```
gcloud builds submit --tag gcr.io/your-project-id/image-name .
```



Cloud Build	Build history				
History	Build history				
Triggers	Build	Source	Git commit	Trigger	Started
	✓ e691570e-8014...	—	—	—	1/24/19, 3:59 PM
	✓ c2083675-d5f1...	Cloud Source Repository pets-app	1b1984a	Push to master branch	1/6/19, 8:35 PM
	✓ ffe4b758-8a4d...	gs://doug-rehnstrom_cloudbuild/source/1546824544.29-edfbdfcc50e514eb87493e3b48cbcc9.tgz	—	—	1/6/19, 8:29 PM
	! aced01e1-2d22...	gs://doug-rehnstrom_cloudbuild/source/1546824317.76-	—	—	1/6/19, 8:25 PM

Google Cloud Build Triggers

- Watch a repository and build a container whenever code is pushed
 - Works with GitHub and other external git repos
 - Not limited to Docker builds, can use Maven or custom builds



The image shows three sequential steps in the Google Cloud Build Trigger creation process:

- Create trigger** (Step 1: Select source)
Choose a repository hosting option:
 - Cloud Source Repository
 - GitHub
 - Bitbucket

Continue **Cancel**
- Create trigger** (Step 2: Select repository)
Source: Cloud Source Repository
Select repository
Filter repositories:
 default
Cloud Source Repository
Continue **Cancel**
- Create trigger** (Step 3: Trigger settings)
Source: Cloud Source Repository Repository: <https://source.developers.google.com/p/remote>
Trigger settings
Name (Optional): Build My Docker Container
Trigger type:
 - Branch
 - Tag

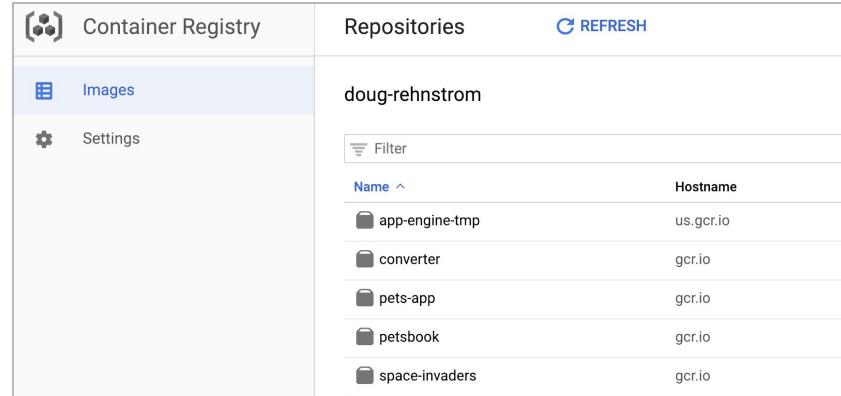
Branch (regex): Matches the branch: master
. *

Build configuration:
 - Dockerfile
Specify the path within the Git repo
 - cloudbuild.yaml
Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Dockerfile directory (Optional): /
The directory will also be used as the Docker build context

Google Container Registry

- Google hosted Docker repository
 - Tag images with the prefix **gcr.io/your-project-id/image-name**
- Can use Docker push and pull commands with Container Registry
 - `docker push gcr.io/your-project-id/image-name`
 - `docker pull gcr.io/your-project-id/image-name`



The screenshot shows the Google Container Registry web interface. On the left is a sidebar with 'Container Registry' at the top, followed by 'Images' (which is selected and highlighted in blue) and 'Settings'. The main area is titled 'Repositories' with a 'REFRESH' button. Below is a table listing five repositories:

Name	Hostname
app-engine-tmp	us.gcr.io
converter	gcr.io
pets-app	gcr.io
petsbook	gcr.io
space-invaders	gcr.io

Tutorials: Google Cloud Build

Do the following tutorials if you want to learn about Google Cloud Build:

- [Quickstart: Build | Cloud Build Documentation](#)
- [Creating and managing build triggers | Cloud Build Documentation](#)

Chapter Concepts

Automated Builds

Jenkins

Jenkins

- Open-source automation server for managing build and deployment tasks
 - <https://jenkins.io/>
 - Is a Java Web application
- Create a pipeline which defines the workflow of a build
 - Monitor source control
 - Check out latest source
 - Build
 - Run tests
 - Deploy
 - Report on results

Creating Jenkins Pipelines

- From the Jenkins home page, select the **New Item** link
 - Give your pipeline a name

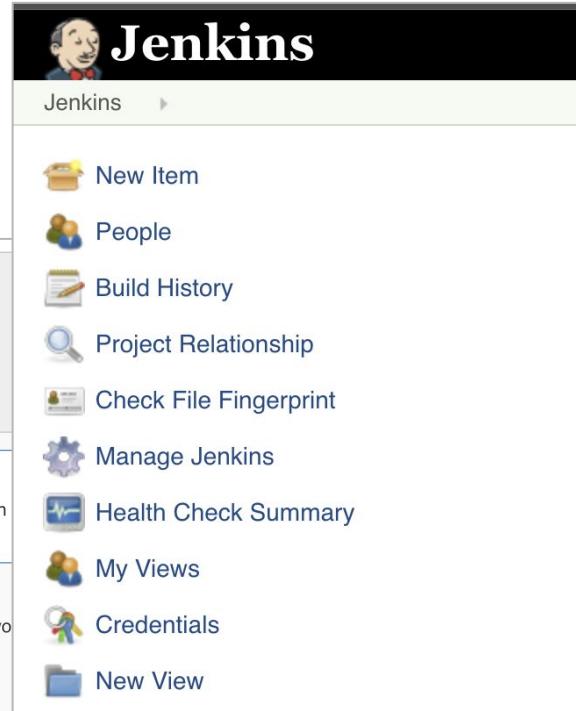
Enter an item name

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as wo organizing complex activities that do not easily fit in free-style job type).

External Job



Jenkins Pipeline Types

The most flexible and configurable option



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Most automated, assuming you are using Maven



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Use this for scripted and declarative pipelines

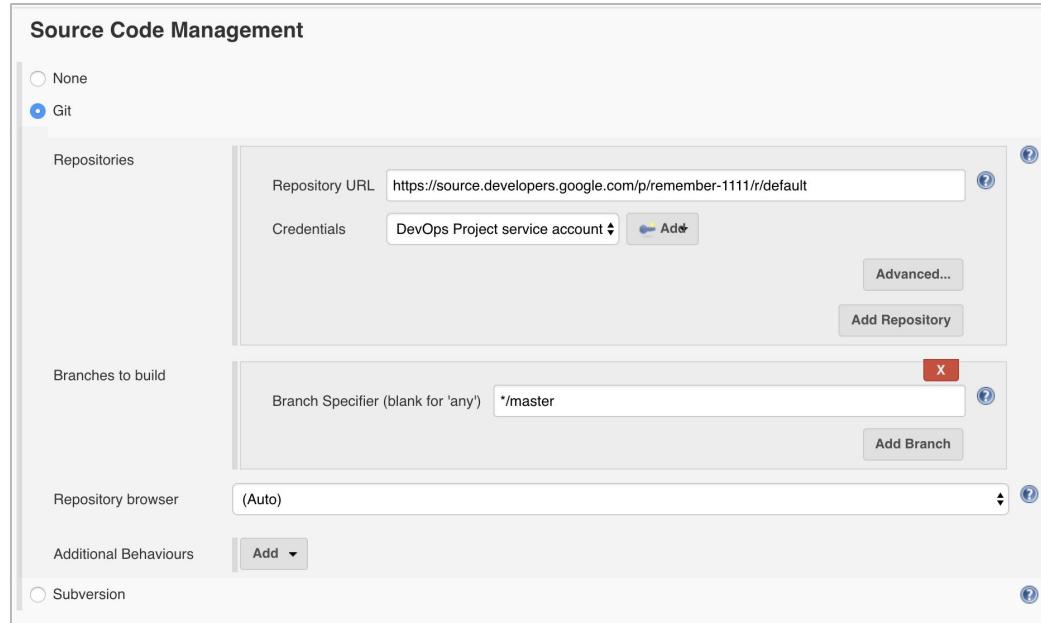


Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Source Code Management

- Git and Subversion are supported
 - Must include authentication information to access private repo



Build Triggers

Build Triggers

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?
- Build periodically ?
- GitHub hook trigger for GITScm polling ?
- Poll SCM ?

Schedule

H * * * *

This would poll once
an hour

Would last have run at Saturday, December 30, 2017 2:20:12 PM UTC; would next run at Saturday, December 30, 2017 3:20:12 PM UTC.

Ignore post-commit hooks



Customizing Build Environment

Build Environment

Delete workspace before build starts

Provide Configuration files

Abort the build if it's stuck

Time-out strategy: Absolute ?

Timeout minutes: 3 ?

Time-out variable:
Set a build timeout environment variable

Time-out actions: Add action ▾ ?

Add timestamps to the Console Output

Google Cloud Ephemeral Deployer

JClouds Single-use slave

Log Build Status to Git Notes

Setup Kubernetes CLI (kubectl) ?

Use secret text(s) or file(s) ?

With Ant ?

Post-Build Steps

Post-build Actions

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like \$BUILD_URL\$ or \$PROJECT_NAME\$. Triggers a post-build action when the build fails, becomes unstable or returns to stable.

- Send e-mail for every unstable build
- Send separate e-mails to individuals who broke the build

Add post-build action ▾

Aggregate downstream test results

Archive the artifacts

Build other projects

Google Cloud Storage Plugin

Publish JUnit test result report

Publish job status to Google Calendar

Record fingerprints of files to track usage

Upload Android APK to Google Play

Git Publisher

Google Cloud Deployer

E-mail Notification

Editable Email Notification

Notify Android devices

Set GitHub commit status (universal)

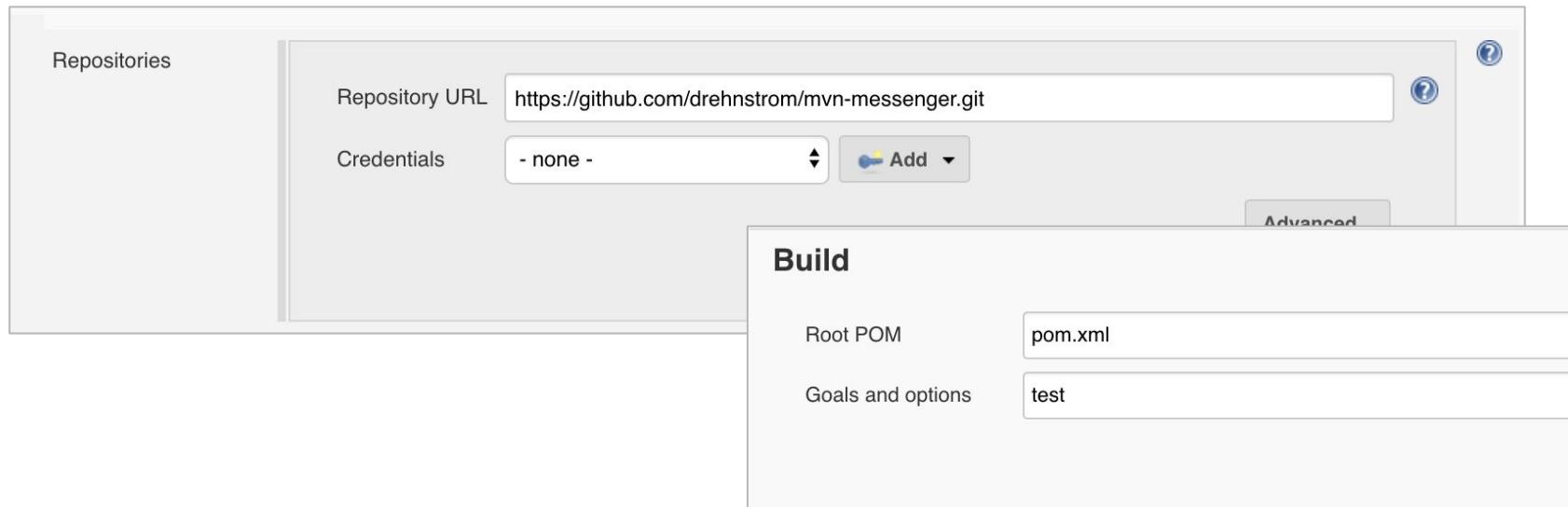
Set build status on GitHub commit [deprecated]

Delete workspace when build is done

Add post-build action ▾

Maven Projects

- Uses the Maven pom.xml file to automate the build
 - Specify the source code repository and trigger
 - Specify the Maven goal



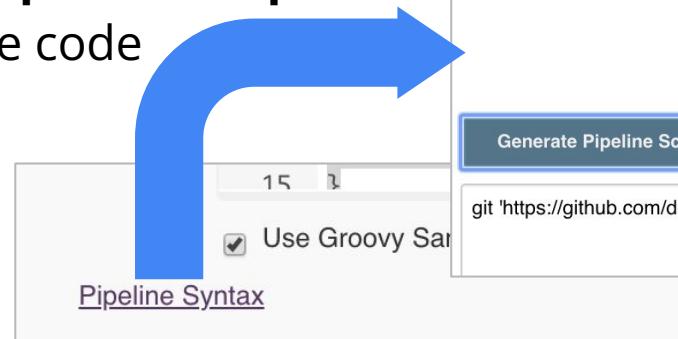
Scripted Pipelines

- Uses Groovy script to define the pipelines
 - To learn about Groovy see: <http://www.groovy-lang.org/index.html>

```
node {  
    stage('Source Code') {  
        echo 'Getting Source Code'  
        // Get code from a GitHub repository  
        git 'https://github.com/drehnstrom/devops-converter.git'  
        sh 'ls -a'  
    }  
    stage('Run the Unit Tests') {  
        sh 'python3 -m pytest'  
    }  
    stage('Build the Docker Image') {  
        sh 'docker build -t gcr.io/doug-rehnstrom/jenkins-converter:v0.2 .'  
        sh 'docker images'  
    }  
}
```

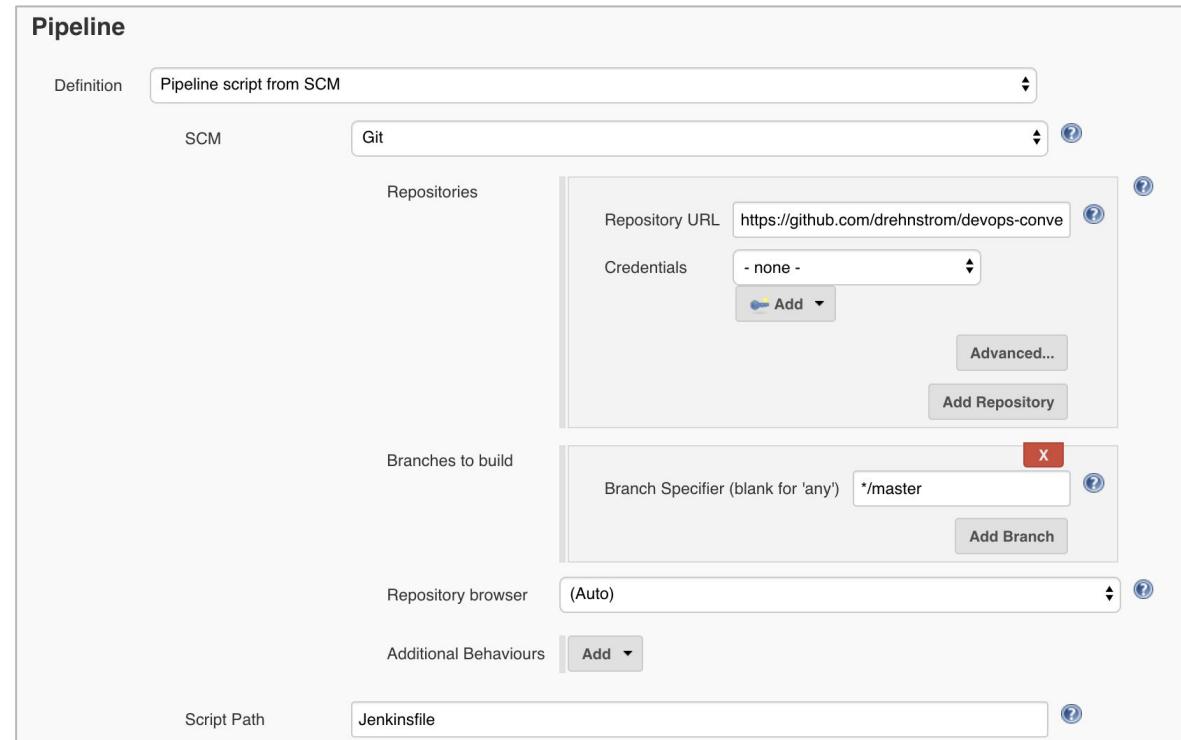
Generating Scripts

- Jenkins provides a UI tool to help generate Groovy scripts from Freestyle
 - Click the **Pipeline Syntax** link
 - Fill in the resulting form
 - Click **Generate Pipeline Script**
 - It will give you the code



Declarative Pipelines

- Groovy script is stored in a file named Jenkinsfile added to the Git repository



Jenkinsfile Example

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
                sh 'pip3 install -r requirements.txt'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
                sh 'python3 -m pytest'
            }
        }
        stage('Package') {
            ...
        }
    }
}
```

Debugging Jenkins Pipelines

- Each pipeline build history is maintained
 - Console output is stored and available during and after a build
 - Examine the console output to see results and error messages

The screenshot shows the Jenkins interface for a pipeline project named "find". On the left, the "Build History" sidebar lists seven builds, each with a blue circular icon and a build number (e.g., #26, #25, #24, #23, #22, #21, #20) followed by the build date and time. A context menu is open over build #21, showing options: Back to Project, Status, Changes, Console Output (which is highlighted in blue), View as plain text, Edit Build Information, Delete build '#21', Git Build Data, and No Tags. The main content area on the right is titled "Console Output" and displays the log for build #21. The log starts with "Started by user admin" and continues with the Jenkinsfile being obtained from git, the pipeline running at MAX_SURVIVABILITY, and the start of the pipeline node.

Build	Date
#26	Aug 11, 2019 3:15 PM
#25	Aug 11, 2019 3:10 PM
#24	Aug 10, 2019 4:12 PM
#23	Aug 9, 2019 4:03 PM
#22	Aug 9, 2019 4:01 PM
#21	Aug 9, 2019 3:47 PM
#20	Aug 9, 2019 3:43 PM



Console Output

```
Started by user admin
Obtained Jenkinsfile from git https://github.com/drehnstrom/
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pipeline-8
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
```

Jenkins and Docker

- Jenkins is available as a Docker image
 - Runs entirely inside a Docker container
- Alternatively, can use Docker images to host build stages
 - No longer need to install multiple languages and tools on base machine
 - Just Docker and Jenkins
 - All other languages frameworks encapsulated inside Docker images
 - Can use existing images or Dockerfiles directly
 - DevOps teams can curate images for this purpose
- Two approaches can be combined
 - Complex to manage permissions securely for Docker-in-Docker

Jenkinsfile with Docker Example

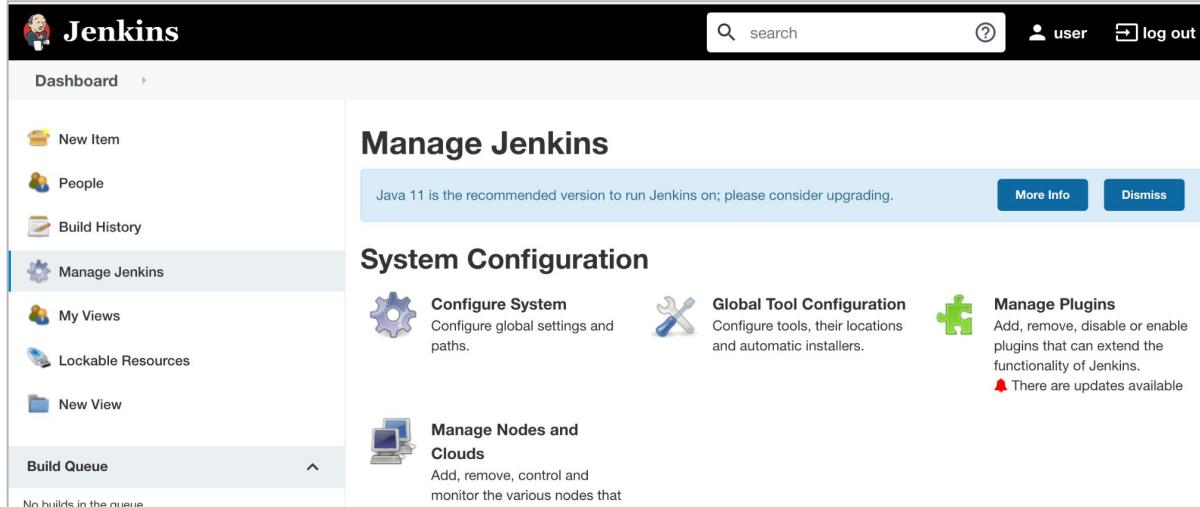
```
pipeline {  
    agent none  
    stages {  
        stage('Run the tests') {  
            agent {  
                docker {  
                    image 'node:14-alpine'  
                    // arg can pass in ENV vars to alias directories  
                    // prevents permissions error writing to disk  
                    args '-e HOME=/tmp -e NPM_CONFIG_PREFIX=/tmp/.npm'  
                }  
            }  
            steps {  
                // code runs inside docker container  
                // no need to install Nodejs on Jenkins machine  
            }  
        }  
    }  
}
```

Jenkinsfile: Docker and kubectl

```
stages {  
    stage('Run the tests') { // code ommitted }  
    stage('Building image') { // docker code ommitted }  
    stage('Get credentials') { // cloud provider code ommitted }  
    stage('Deploy to k8s') {  
        agent { // code ommitted }  
        steps {  
            echo 'use kubectl set image to update image for container'  
            sh '''  
                kubectl set image deployment/demo-ui \  
                demo-ui=gcr.io/proj-id/image:v1.${env.BUILD_ID} --record  
            '''  
        }  
    }  
    stage('Remove local docker images') { // cleanup code ommitted }  
}
```

Docker Pipeline Extension

- Provides integration with Docker
 - E.g., ability to push to Docker Hub
- Is added inside **Manage Jenkins | Manage Plugins**



Docker Hub Credentials

- Are added as a global credential
 - ID of credential is then supplied to the pipeline
- Credential/login details added inside **Manage Jenkins | Manage Credentials**

The screenshot shows two Jenkins management pages. The top page is titled 'Credentials' and lists a single global credential. The bottom page is titled 'Stores scoped to Jenkins' and shows the Jenkins store with a global domain.

T	P	Store ↓	Domain	ID	Name
		Jenkins	(global)	ff809ea2-4ccd-4c12-9cbb-ebd111dfa3f5	krattan/******** (docker)

Icon: S M L

P	Store ↓	Domains
	Jenkins	!(global)

Jenkinsfile: Using the Docker Plugin

```
pipeline {  
    agent any  
    environment {  
        // docker credentials must be created first. Id defaults to guid  
        registryCredential = 'ff809ea2-4ccd-4c12-9cbb-ebd11dfa3f5'  
        // imageName and dockerImage are variables that can be referenced  
        // inside the steps  
        imageName = 'yourdockerid/yourappname'  
        dockerImage = ''  
    }  
    stages {  
        stage('Run the tests') { //code ommitted }  
        stage('Building image') { //code ommitted }  
        stage('Get credentials') { //code ommitted }  
        stage('Deploy to k8s') { //code ommitted }  
        stage('Remove local docker images') { //code ommitted }  
    }  
}
```

Using the Docker Plugin

- The Docker plugin provides methods to build and push images
- Add `script {}` sections inside steps

```
steps {  
    script {  
        dockerImage = docker.build imageName  
    }  
}  
steps{  
    script {  
        docker.withRegistry( '', registryCredential ) {  
            dockerImage.push("$BUILD_NUMBER") }  
            // $BUILD_NUMBER automatically provided by Jenkins  
    }  
}
```

Combining Docker Plugin with Docker Agents

- Make sure Docker plugin is sharing workspace with Docker agent stages
 - Add reuseNode flag to all Docker agents

```
agent {  
    docker {  
        image 'google/cloud-sdk:latest'  
        reuseNode true  
    }  
}
```

Tips for Getting Pipelines Working

- Carefully plan the steps and what happens in each step
- Work on one step at a time
- Use the Console output to see error messages; common errors include:
 - Authorization errors
 - Command not found errors
 - Syntax errors

Qwiklabs: CI/CD

- [Setting up Jenkins on Kubernetes Engine](#)
- [Continuous Delivery with Jenkins in Kubernetes Engine](#)
- [Getting Started with DevOps on AWS](#)

Activity: Building a CI/CD Pipeline

- In your teams, set up a CI/CD pipeline for your case study using Jenkins
 - Make a change to your source code and push it to your repo
 - This should trigger the Jenkins pipeline to build your application and deploy it
- Or perform one of the Qwiklabs on the topic

Final Activity: Cleaning Up Cloud Resources

- In the cloud, you pay for what you provision
 - Extremely important to delete resources when no longer needed
 - Or you may be justifying why your spend is so high

Chapter Summary

In this chapter, you have:

- Automated builds and deployments using CI/CD pipelines
- Built a CI/CD pipeline with Jenkins

Quiz

What event usually causes a CI/CD pipeline to start?

- A. Running a Maven build
- B. A developer changes the code
- C. A new Docker image being built
- D. Code being pushed to a repository

Quiz

Of the following, what might be included in a continuous integration pipeline?

- A. Compile code
- B. Run automated tests
- C. Run code quality analysis
- D. Create a Docker image or deployment package
- E. All of the above

Quiz

Which tool can be used to help build automated CI/CD pipelines?

- A. AWS CodeBuild and AWS CodePipeline
- B. Google Cloud Build
- C. Jenkins
- D. All of the above



Course Summary

Course Summary

In this course, you have learned how to:

- Design, architect, build, and deploy cloud-native applications
- Choose the right services for public, private, and hybrid cloud deployments
- Program applications using cloud-native best practices
- Optimize cloud resources by leveraging microservice architectures
- Containerize applications using Docker
- Orchestrate container deployment using Kubernetes
- Simplify microservice deployments using PaaS and serverless environments
- Leverage cloud-based data storage services
- Automate builds and deployments using modern DevOps tools

ROI's Training Curricula

- Agile Development
- Amazon Web Services (AWS)
- Azure
- Big Data and Data Analytics
- Business Analysis
- Cloud Computing and Virtualization
- Excel and VBA
- Google Cloud
- ITIL® and IT Service Management
- Java
- Leadership and Management Skills
- Machine Learning and Neural Networks
- Microsoft Exchange
- .NET and Visual Studio
- Networking and IPv6
- Oracle and SQL Server Databases
- OpenStack and Docker
- Project Management
- Python and Perl Programming
- Security
- SharePoint
- Software Analysis and Design
- Software Engineering
- UNIX and Linux
- Web and Mobile Apps
- Windows and Windows Server

Our full [course list](#) is available at www.ROITraining.com

