



# Welcome!

## Deloitte Cloud-Native Development



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



# Introduction

# Course Schedule

- Start of class: \_\_\_\_\_
- Break: \_\_\_\_\_
- Lunch: \_\_\_\_\_
- Resume: \_\_\_\_\_
- Break: \_\_\_\_\_
- Class ends: \_\_\_\_\_

# 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)



# 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 12-Factor App best practices
  - Automate builds, testing, and code analysis in a CI/CD pipeline
  - Use Cloud services whenever possible
  - 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 Google Cloud
  - Though the principles learned can be applied to any cloud provider
    - AWS, Google Cloud, Azure, etc.
- You will be provided with a Google 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
  - 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):
  - Amazon Web Services (AWS)
    - <https://aws.amazon.com/free>
  - Google Cloud
    - <https://cloud.google.com/free>
  - Microsoft Azure
    - <http://azure.microsoft.com/free>

# Why Are We Using Google Cloud?

- Lessons in this course apply to all clouds
  - A Google Cloud account will be provided to use during this course
- We are really just using Google Cloud because of Cloud Shell
  - Preconfigured VM makes setup easier
  - Built-in Code Editor allows labs to be done without configuring tools
  - Integration with Google Cloud avoids some security issues
- This course focuses on open-source and/or cloud agnostic tools
  - Git
  - Docker
  - Kubernetes
  - Terraform

# Logging In to Google Cloud

- Your instructor will explain how to get logged in to Google Cloud

The screenshot shows the Google Cloud Platform dashboard for a project named "My Project 89289".

- Project info:** Shows the project name (My Project 89289), project ID (mindful-primer-272612), and project number (371093259438). It also includes a link to "ADD PEOPLE TO THIS PROJECT" and a "Go to project settings" button.
- App Engine:** Displays a summary chart for count/sec over time (7 AM to 7:45). A message indicates "No data is available for the selected time frame." Below the chart is a link to "Go to the App Engine dashboard".
- Compute Engine:** Shows CPU usage at 0.32%.
- Google Cloud Platform status:** Shows "All services normal" and a link to "Go to Cloud status dashboard".
- Billing:** Shows estimated charges of USD \$24.87 for the billing period April 1 – 13, 2020. A link to "View detailed charges" is provided.
- Error Reporting:** Shows top errors in the last 24 hours, including 725 TypeErrors related to "get()(server.js)". A link to "Go to Error Reporting" is provided.

# Google Cloud Shell

- Online Code Editor provided free as a part of Google Cloud
  - 5GB of storage per account
- Linux VM has requirements for this course pre-installed
  - Node.js
  - Python
  - Java
  - Docker
  - Terraform
  - Git
  - Etc.

The screenshot shows the Google Cloud Shell interface. On the left, there's a file explorer window titled 'EXPLORER: DREHNSTROM' containing files like 'status', 'templates', 'app-flex.yaml', 'app.yaml', 'appengine\_config.py', 'Cheat\_Sheet.txt', 'deployment-manager-config.yaml', 'Dockerfile', 'kubernetes-config.yaml', and 'main.py'. The 'Dockerfile' tab is selected, displaying the following code:

```
FROM python:3.7
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
RUN pip install gunicorn
ENV PORT=8080
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 main:app
```

Below the file explorer is a terminal window with the following text:

```
Welcome to Cloud Shell! Type "help" to get started.  
Your Cloud Platform project in this session is set to mindful-primer-272612.  
Use "gcloud config set project [PROJECT ID]" to change to a different project.  
drehnstrom@cloudshell:~ (mindful-primer-272612) $
```

# Qwiklabs

- During the course, we will also provide Qwiklabs to allow you to gain more hands-on experience
  - You will be provided with Qwiklab credits to use
  - These credits are for you to use outside of class hours to gain more knowledge on a topic
    - Generally, the Qwiklabs will not be done in class
- You will need to create a free Qwiklab account:
  - Go to <https://www.qwiklabs.com/>
  - Click the **Join** button
  - Follow the prompts to create new account



# 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

# Agenda

## 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](http://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? \_\_\_\_\_

# Agenda

Defining Cloud Computing

---

## **Cloud Deployment Models**

---

Cloud Platforms

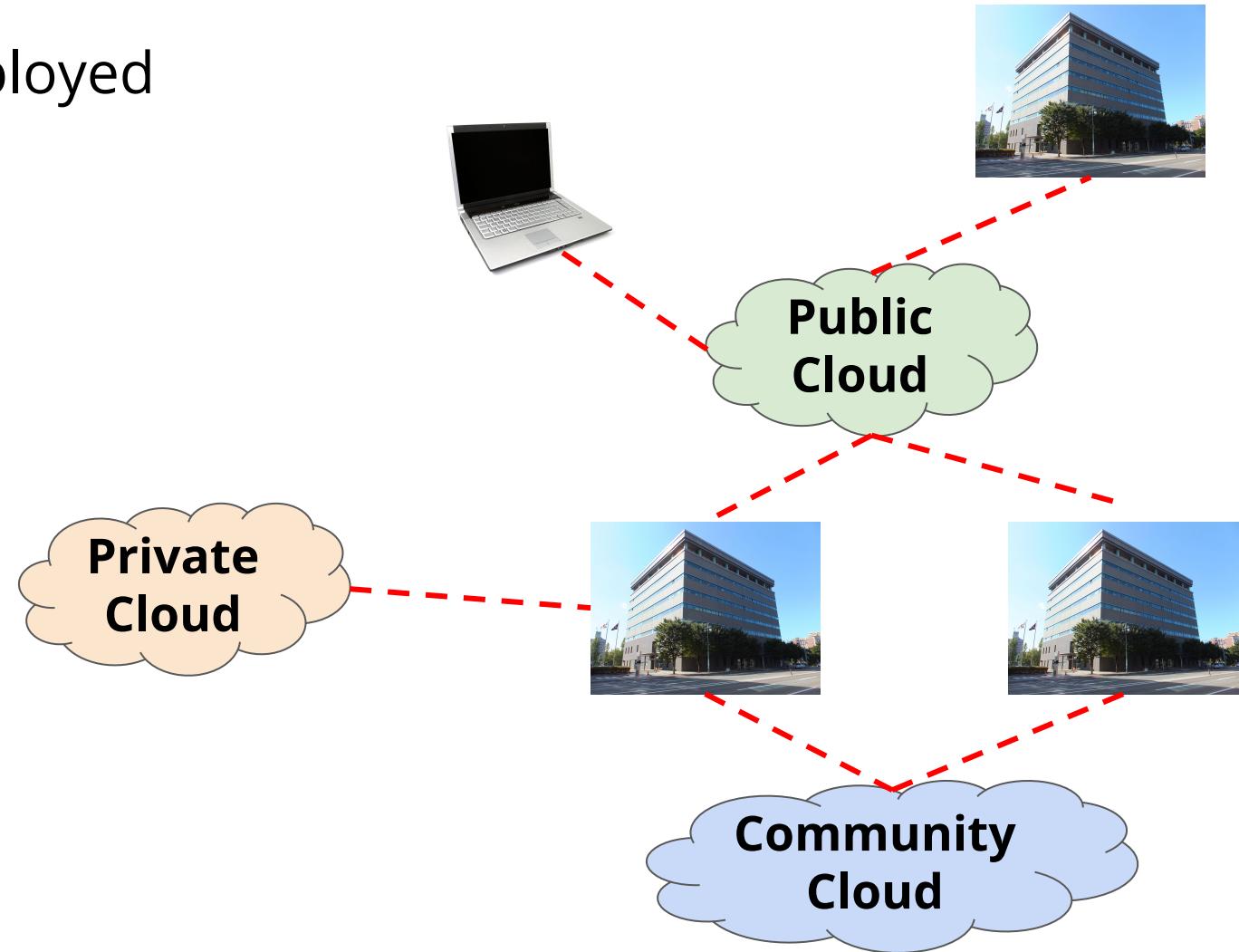
---

Cloud Service Models

---

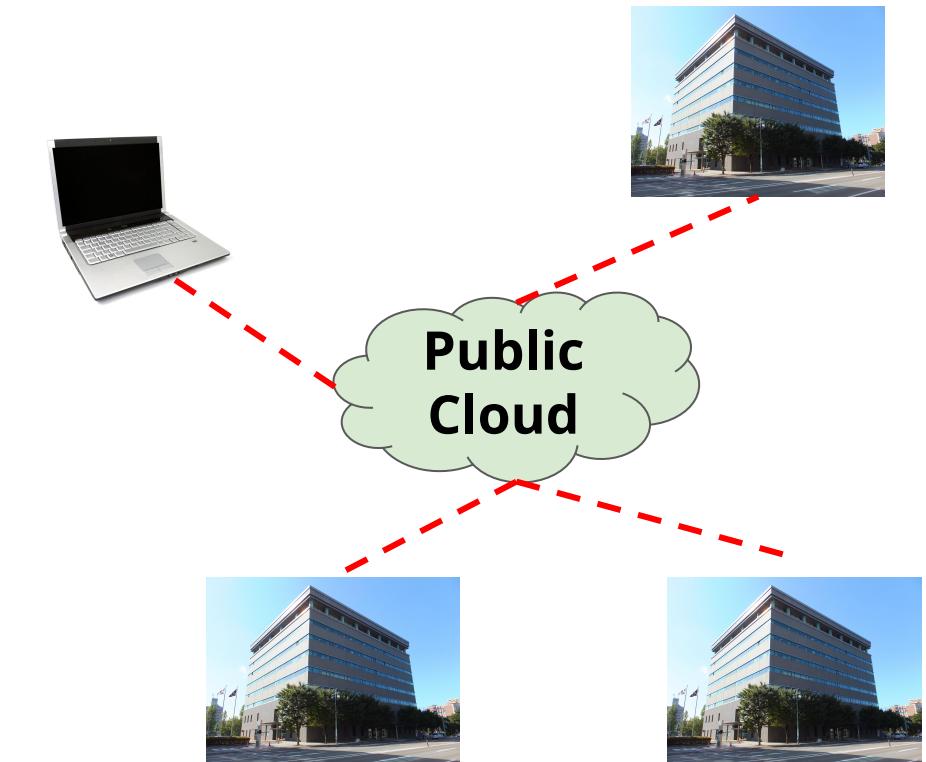
# Deployment Models

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



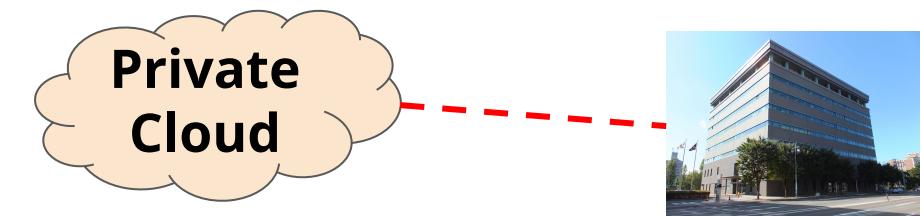
# Public Cloud

- Cloud services hosted by a third-party organization
  - And made available to the general public
  - A multi-tenant environment
- Main public cloud providers:
  - Amazon Web Services (AWS)
  - Microsoft Azure
  - Google Cloud
  - IBM
  - Alibaba
  - DigitalOcean
  - SkyTap
  - Etc.



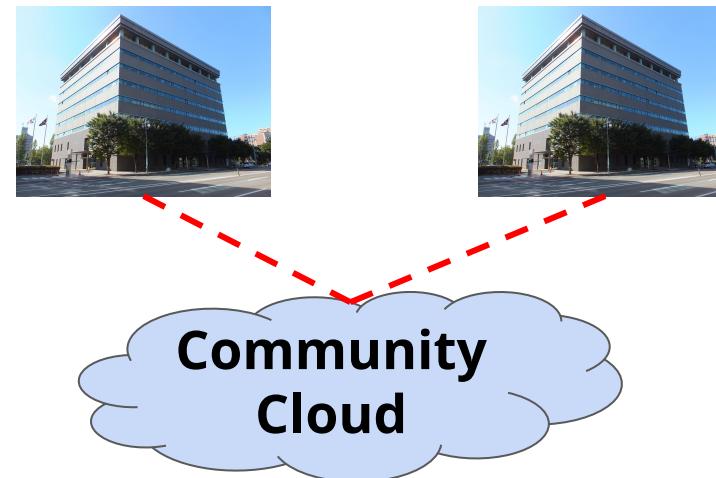
# Private Cloud

- Cloud services dedicated to a single organization
  - Single-tenant
  - May be easier to comply with corporate security standards, policies, and regulatory requirements
- On-premises private cloud
  - Managed by organization or third party
- Hosted private cloud
  - Managed by third party
- Some cloud management software examples:
  - Openstack
  - VMWare VCloud
  - Microsoft System Center



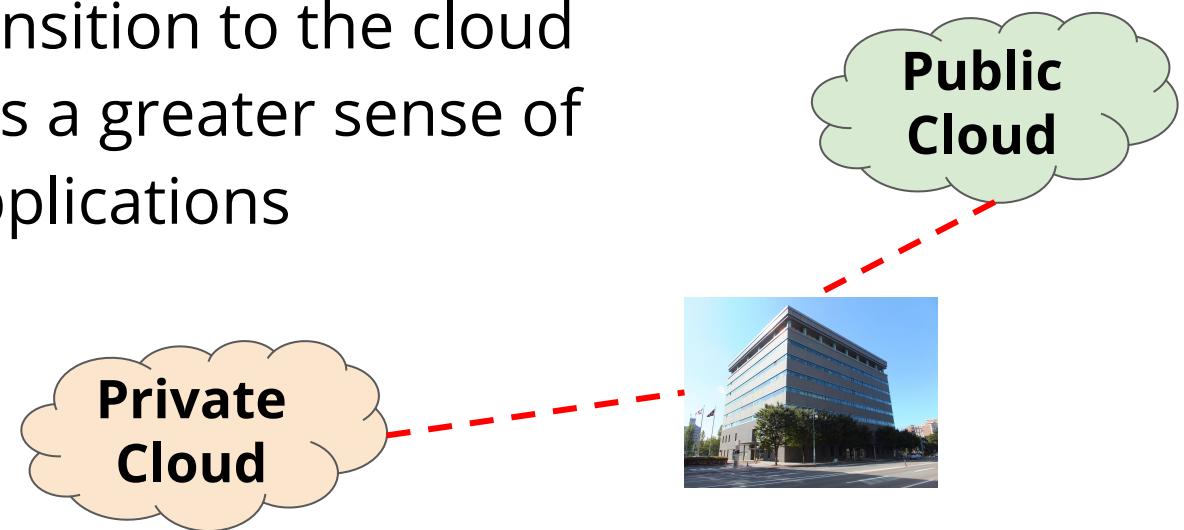
# Community Cloud

- Cloud services shared among several organizations
  - Organizations must all be in agreement and have common needs
  - Not available to the general public
- Can be hosted by organizations or by third party
  - Each participant organization may provide cloud services, consume cloud services, or both



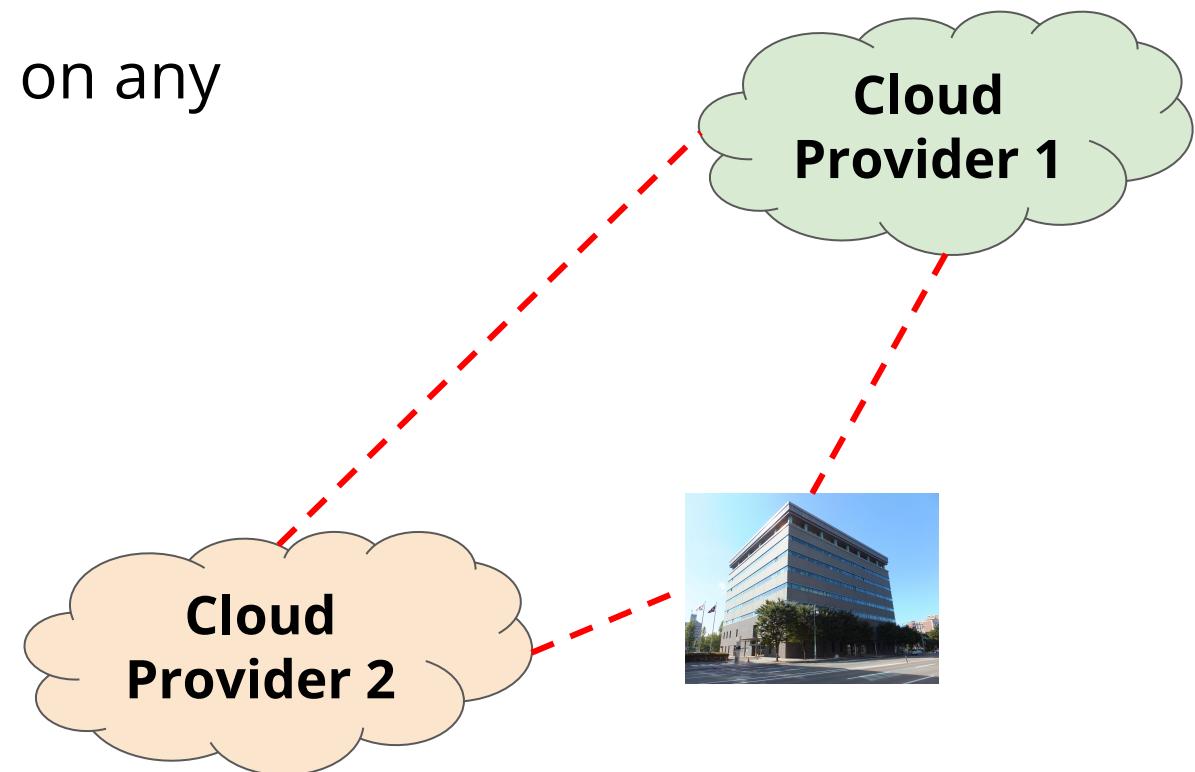
# Hybrid Cloud

- Combination of two or more cloud deployment models
  - Leverage the best of all models
- Hybrid clouds can link on-premises and public clouds together
  - Some data stays where it is and some is moved
  - Applications link to each other between local and cloud data centers
- Hybrid clouds offer a smoother transition to the cloud
  - Hybrid clouds give organizations a greater sense of control over critical data and applications



# Multi Cloud

- Integrate two or more cloud providers
  - In a single heterogeneous architecture
- Goal is to eliminate the reliance on any single cloud provider



# Agenda

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)**
  - **Microsoft Azure**
  - **Google Cloud**
  - IBM Cloud
  - Alibaba Cloud
- AWS, Azure, and Google Cloud are usually considered the major players

# Amazon Web Services (AWS)

- One of the oldest public cloud platforms
  - Publically available since March 2006

The screenshot shows the AWS Management Console homepage. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, a star icon, a notification bell with a blue dot, 'Cloud Native' dropdown, 'N. Virginia' dropdown, and 'Support' dropdown. The main title 'AWS Management Console' is centered above two columns of content.

**AWS services**

**Find Services**  
You can enter names, keywords or acronyms.  
 Example: Relational Database Service, database, RDS

▼ Recently visited services  
 IAM       CloudFormation

**Access resources on the go**  
 Access the Management Console using the AWS Console Mobile App. [Learn more](#)

**Explore AWS**

# Google Cloud

The screenshot shows the Google Cloud Platform dashboard with the following sections:

- Project info:** Project name: CloudNative, Project ID: cloudnative-247610, Project number: 381191131231. A button to "Go to project settings".
- APIs:** Requests (requests/sec) chart from 5:30 to 6:15. The chart shows a value of 0.6. A note: "No data is available for the selected time frame."
- Google Cloud Platform status:** All services normal. A link to "Go to Cloud status dashboard".
- Billing:** Estimated charges USD \$0.00 for the billing period Jul 1 – 23, 2019. A link to "View detailed charges".

# Microsoft Azure

Microsoft Azure  Search resources, services, and docs     ? 

« Create a resource 

Home 

Dashboard 

All services 

★ FAVORITES

Resource groups 

All resources 

Recent 

App Services 

Azure services  See all (100+)  Create a resource >

 Virtual machines  App Services  Storage accounts  SQL databases

 Azure Cosmos DB  Kubernetes services  Function App

 Microsoft Learn  
Learn Azure with free  Azure Monitor  
Monitor your apps and

# Agenda

Defining Cloud Computing

---

Cloud Deployment Models

---

Cloud Platforms

---

**Cloud Service Models**

---

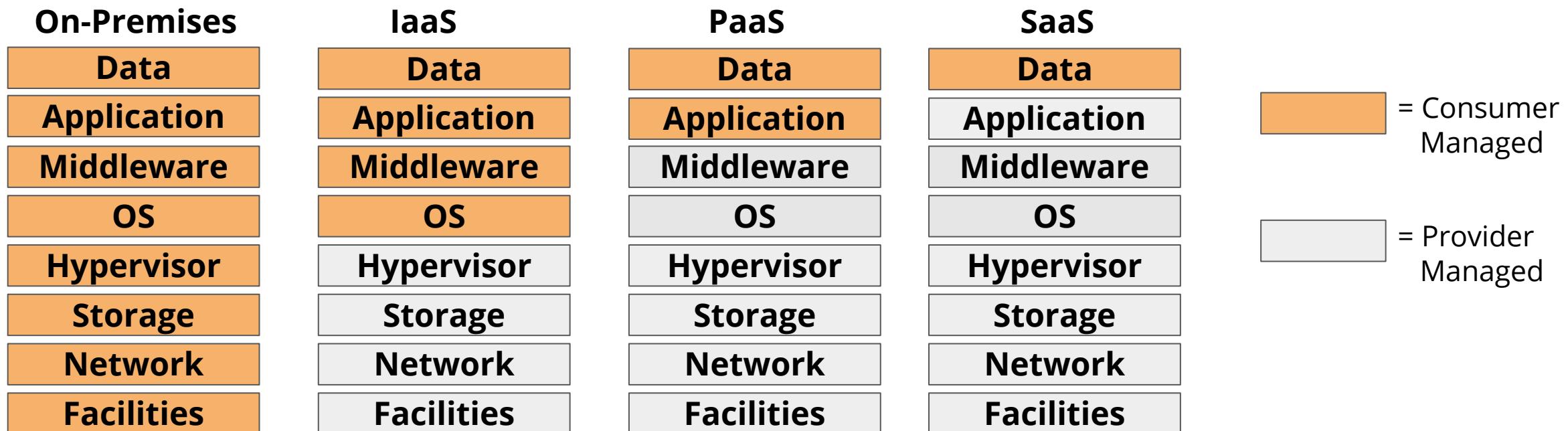
# IT Ops/DevOps/NoOps

- 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
- No Ops 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



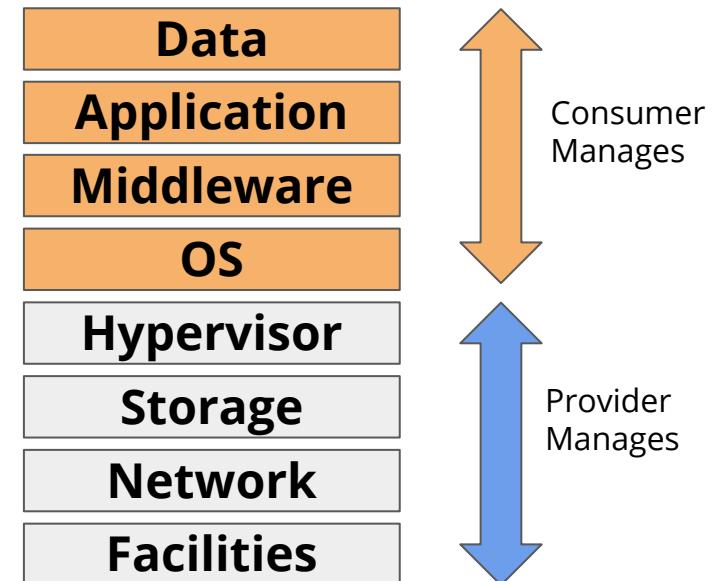
# Cloud Service Models

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



# 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
  - Can configure however you want
  - Can install any software you want
- But consumers are responsible for maintaining it
  - Cloud provider does not maintain, backup, patch, update the software
- Not much different from administering local servers



# IaaS

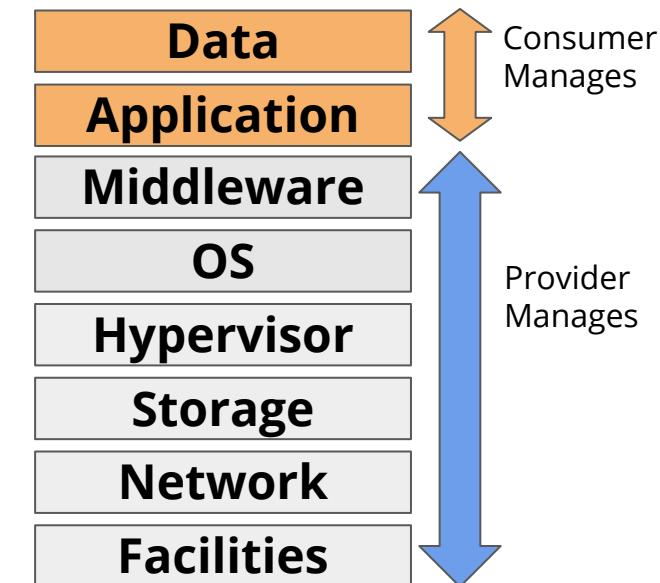
- Often the easiest path onto the cloud
  - Migrate on-premises VMs to VMs hosted in the cloud
- Complete control over operating system, configuration, software
  - You are responsible for all maintenance other than hardware

# Tutorials: Creating a VM in the Cloud

- AWS
  - <https://www.qwiklabs.com/focuses/14761?parent=catalog>
- Google Cloud
  - <https://www.qwiklabs.com/focuses/3563?parent=catalog>
  - <https://www.qwiklabs.com/focuses/560?parent=catalog> (Windows)

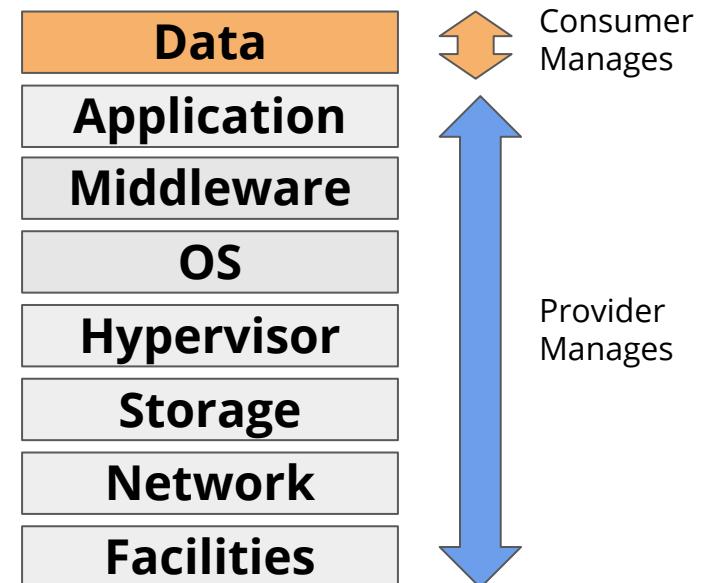
# Platform as a Service (PaaS)

- 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)

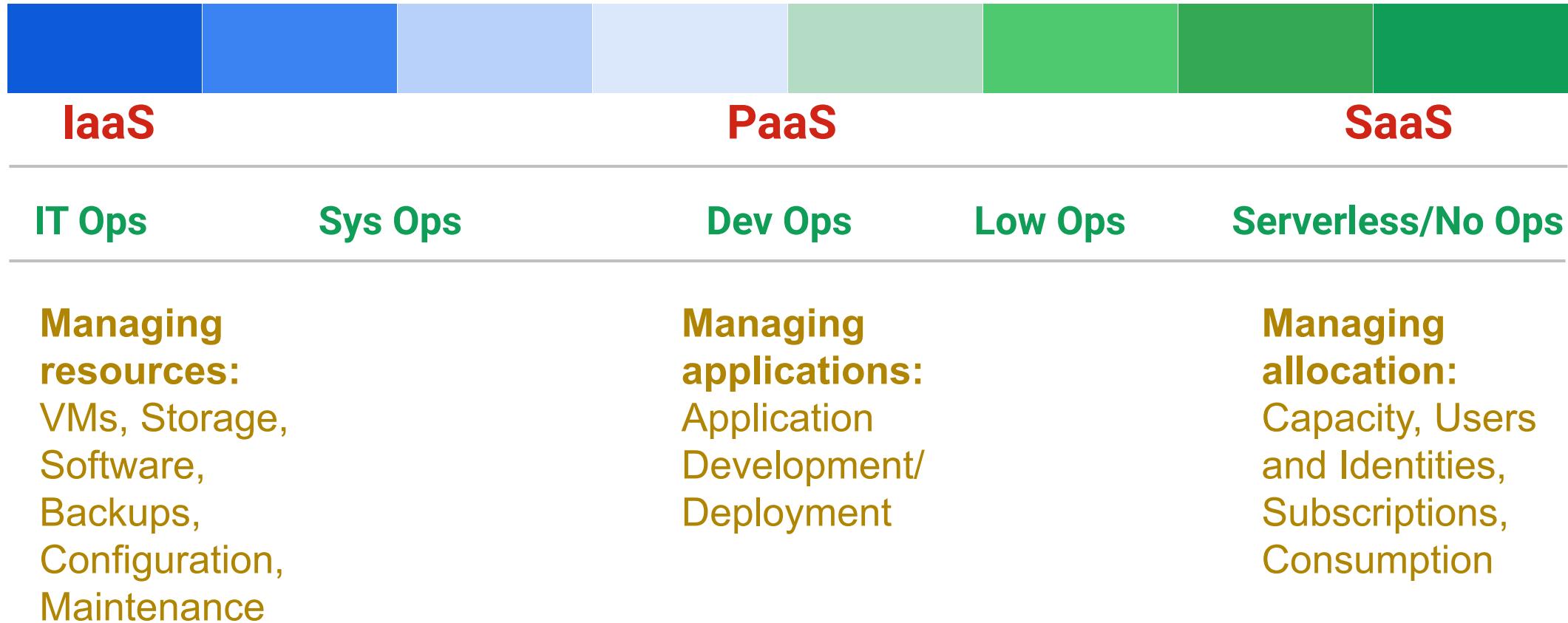
- Model where provider licenses application use as a service on demand
  - No need to manually deploy or manage infrastructure in the cloud
  - No need to install software on the client computer
  - Provider supplies the entire infrastructure and software application
- Many traditional software applications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



# SaaS — Serverless Computing

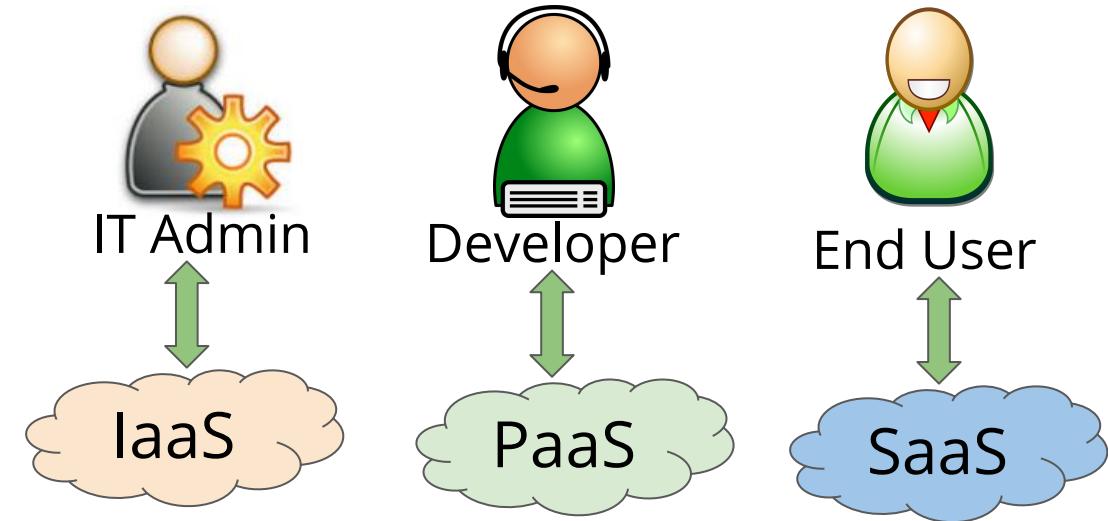
- SaaS can also be used to deliver cloud services
  - Some sophisticated cloud services are consumed as a SaaS no-ops solution
  - Sometimes called *serverless* computing
  - No need to manually provision or manage servers
    - There are still servers running, but the provider dynamically manages the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

# Cloud Service Models (continued)



# Service Model Users

- Originally, the service models catered to specific roles
- The new “serverless” or no-ops services are changing that
  - Cloud providers are developing SaaS that target all job roles
  - AWS Lambda, Azure Functions, and Google Cloud Functions are SaaS for developers
  - BigQuery, Athena, and Data Lake Analytics are for data scientists/business analysts
  - These do not require an admin to provision or manage anything



# Managed Database

- Delivery of a hosted database environment as a service
  - Provides a database service without the need for setting up physical hardware, installing software, or configuring for performance
  - All administrative/maintenance tasks are performed by the provider
  - Often referred to as Database as a Service (DBaaS)
- Cloud providers offer both relational and NoSQL (non-relational) databases as a service

# Demonstration: Creating a Managed Database



- You instructor will now demonstrate creating a managed database in the cloud

# 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
  - Database
  - 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)



# 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

# Agenda

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

# Agenda

Cloud-Native Development

---

## Agile Practices

---

Automation and DevOps

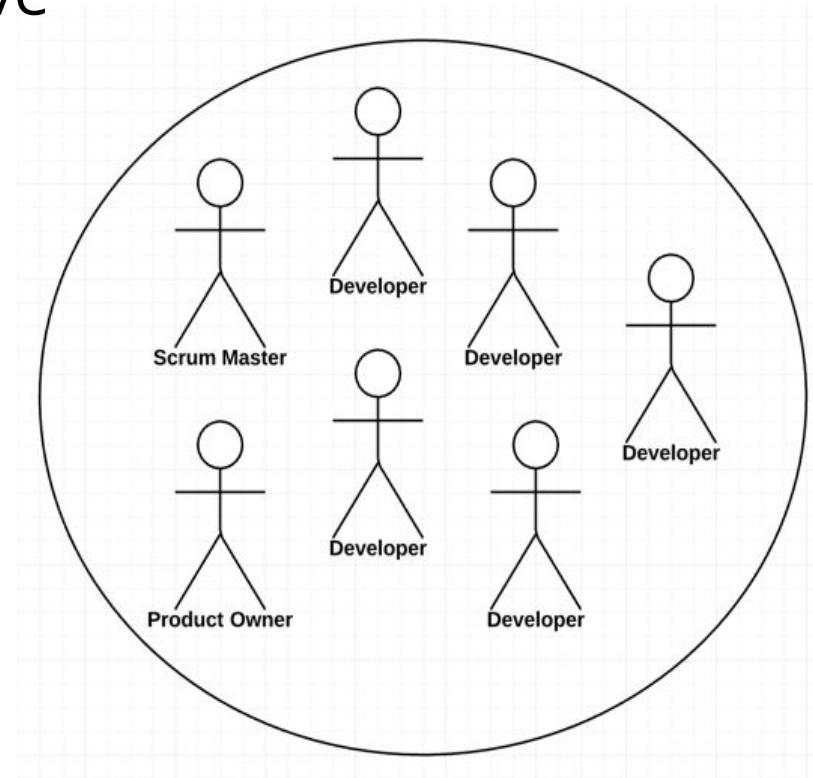
---

# Agile Management Practices

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

# Agile Teams

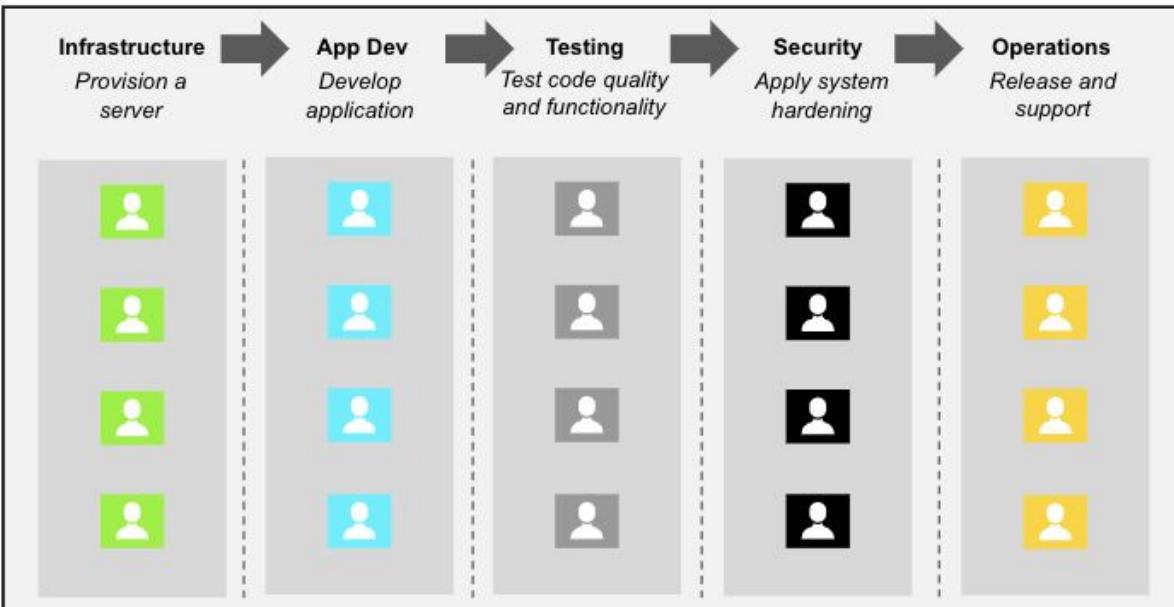
- Consist of a Scrum Master, Product Owner, and a collection of Developers
  - Developers work in a variety of roles to get the software done
- Agile teams are cross-functional and collaborative
- Agile teams require:
  - Cross training
  - People willing to learn new things
  - People with multiple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



# Traditional Teams Are Siloed, DevOps Teams Need to Be Collaborative

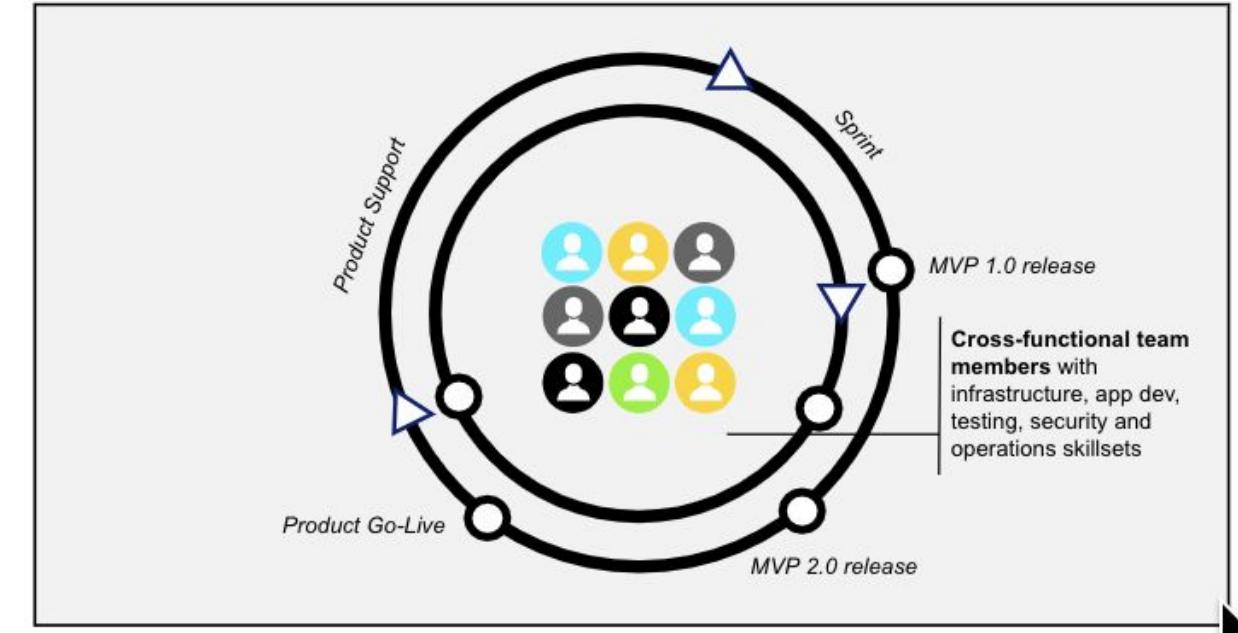
## Traditional Team Structures

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



# Recognizing High-Performing Agile Teams

## Release as a Non-Event

*Small and frequent releases deploying features instead of a big-bang production going live*

## Accelerated Feedback

*Feedback through early testing and real-time monitoring factored in decision making*

## Stop Starting and Start Finishing

*Limit "Work In Progress" to deliver value frequently and incrementally*

## Continuous Improvement

*Continuously challenge the status quo through culture of experimentation and retrospectives*

## Real-Time Communication

*Communicate in real time across the functions for low-latency decisioning*

Habits

## Outcomes

Metrics	High Performers
<b>Leading DevOps KPIs</b>	
Release frequency	On demand (multiple times/day)
Lead time for changes	<1 hour
Mean time to recovery	<1 hour
Change failure rate	0-15%
<b>Test Automation &amp; Coverage</b>	
Unit test - UI code	50% new code
Unit test - services	90% new code
Regression/functional automation	Covers ~80% of user activity
<b>Managing Queues</b>	
Size of Epics	3 months or less
Size of User Stories	1 week or less
Planning commitment horizon	Quarterly or less
Work In Progress limits	1-2 items/engineer

# 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 at least 10 users stories for the course case study
- 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)

# Agenda

Cloud-Native Development

---

Agile Practices

---

**Automation and DevOps**

---

# Automation

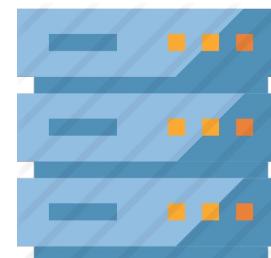
- Software is complex and only getting more so
  - Humans need help automating repetitive, time-consuming tasks
  - Computers are really fast and never make mistakes
  - Computers enjoy repetitive tasks!
- Automation can help with:
  - Tests
  - Builds
  - Deployment
  - Configuration
  - Etc.

# 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
  - Amazon 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 Cloud Formation?**

- 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 12-Factor App best practices
- Understand microservice communication protocols

# Agenda

## Introduction to Microservices

---

12-Factor Apps

---

Microservice Architecture

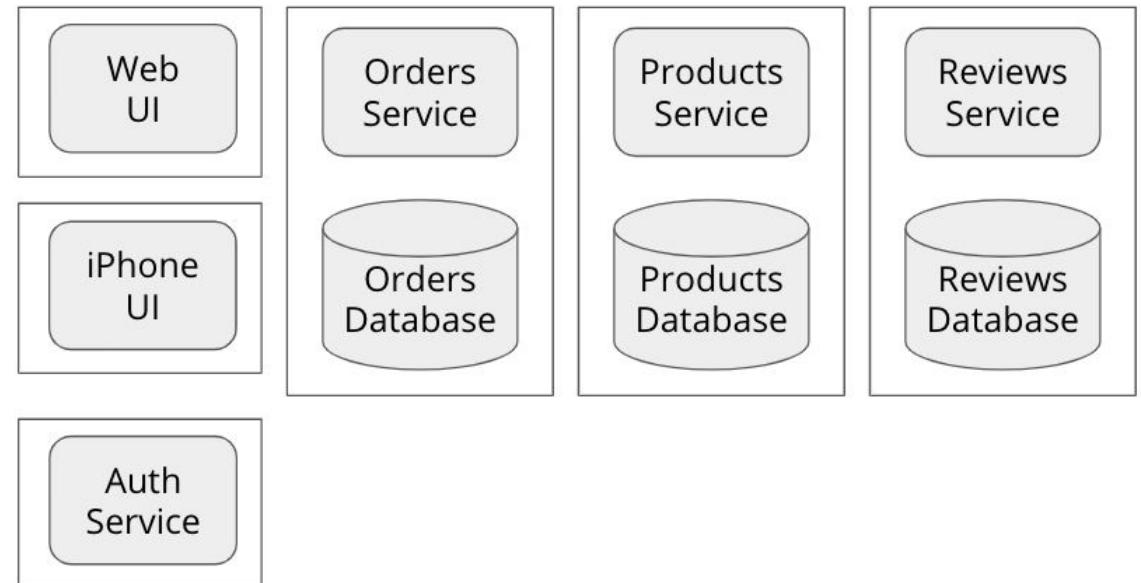
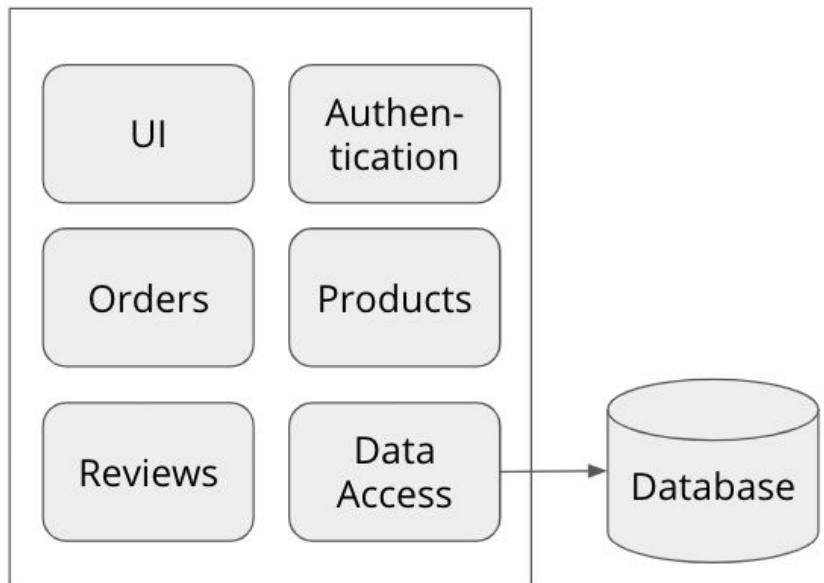
---

# 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

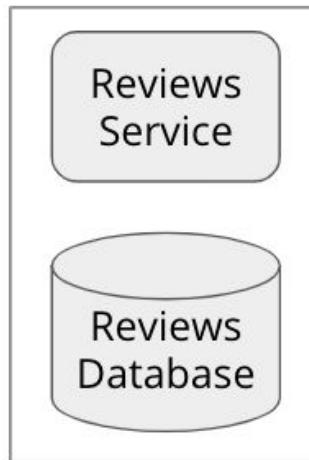


# Recognizing Microservice Boundaries

- 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

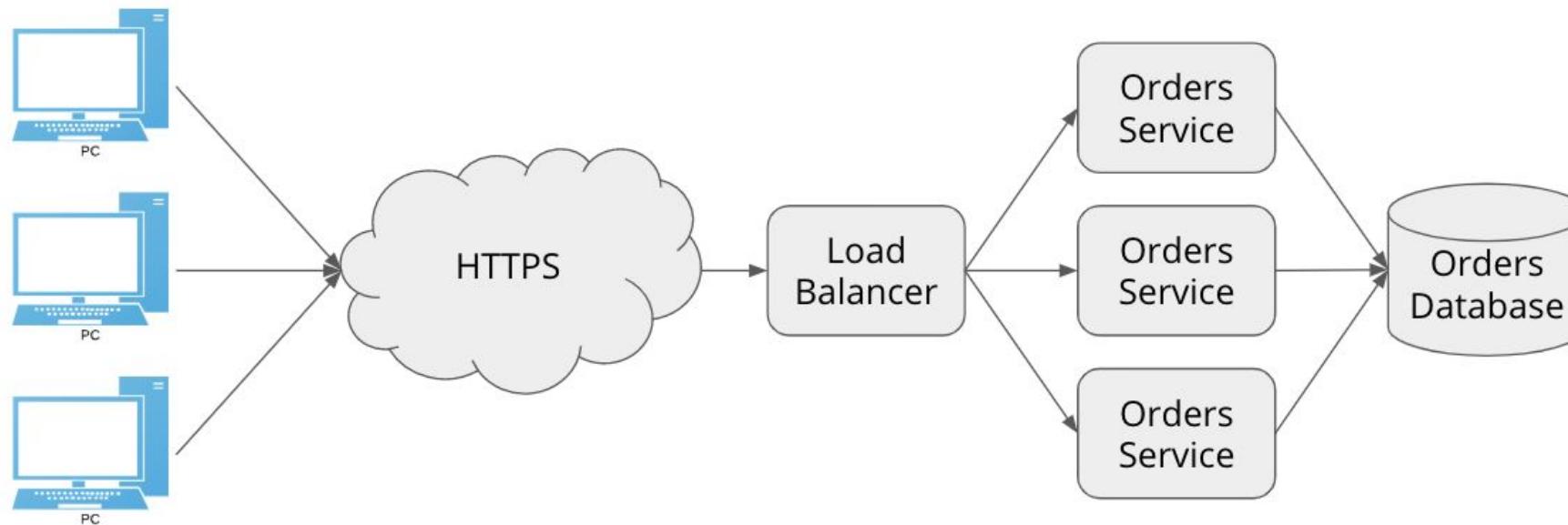
# Stateful vs. Stateless Services

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



# 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, master 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

# Agenda

Introduction to Microservices

---

**12-Factor Apps**

---

Microservice Architecture

---

# 12-Factor Apps

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



## THE TWELVE-FACTOR APP

### INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- 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.

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 12 Factors

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

# The 12 Factors (continued)

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

# The 12 Factors (continued)

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

# Activity: Implementing 12-Factor Apps

- As a group, come up with specific ways you can ensure your case study application conforms to the 12 factors

# Agenda

Introduction to Microservices

---

12-Factor Apps

---

**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
- Something else ...

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

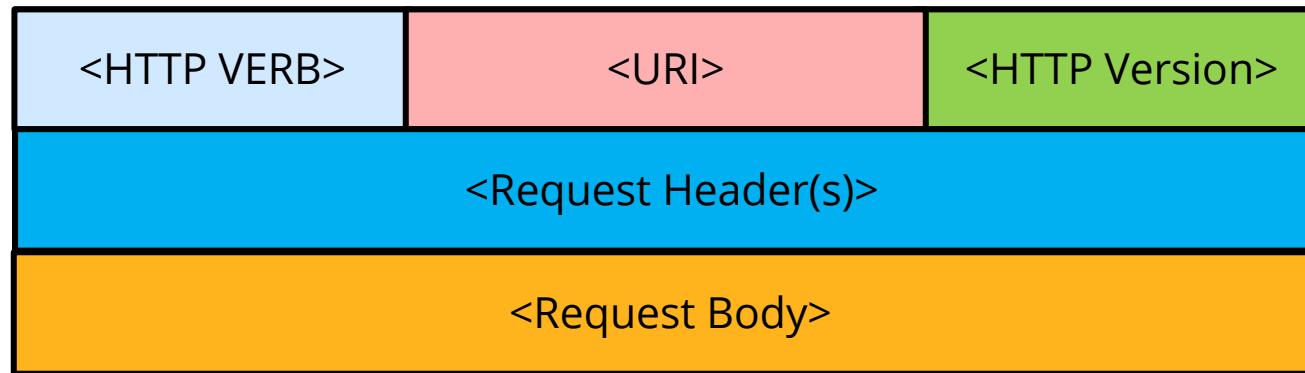
```
{"pets": [
  {"name": "Noir", "breed": "Schnoodle"},  

  {"name": "Bree", "breed": "Mutt"}]
```

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

name, breed  
Noir, Schnoodle  
Bree, 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

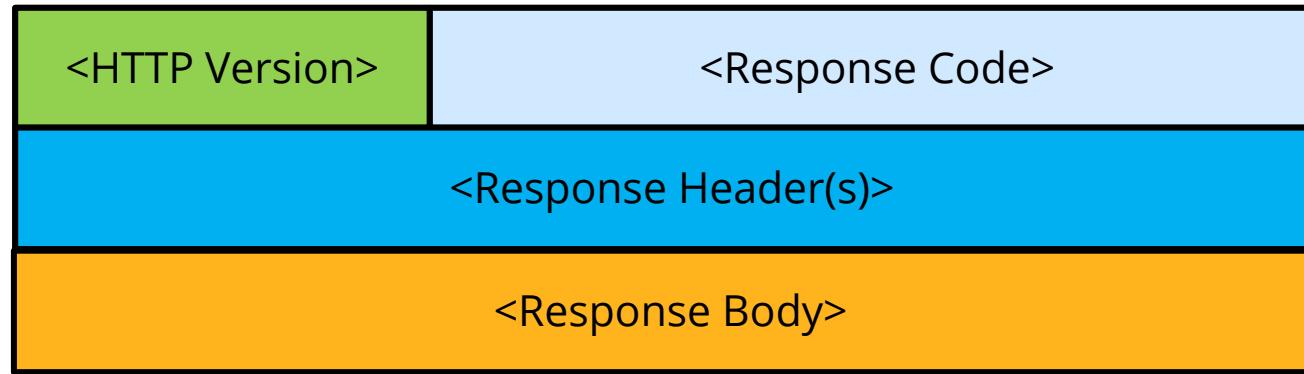
- GET

```
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](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)
- Response Body: Contains Resource Representation

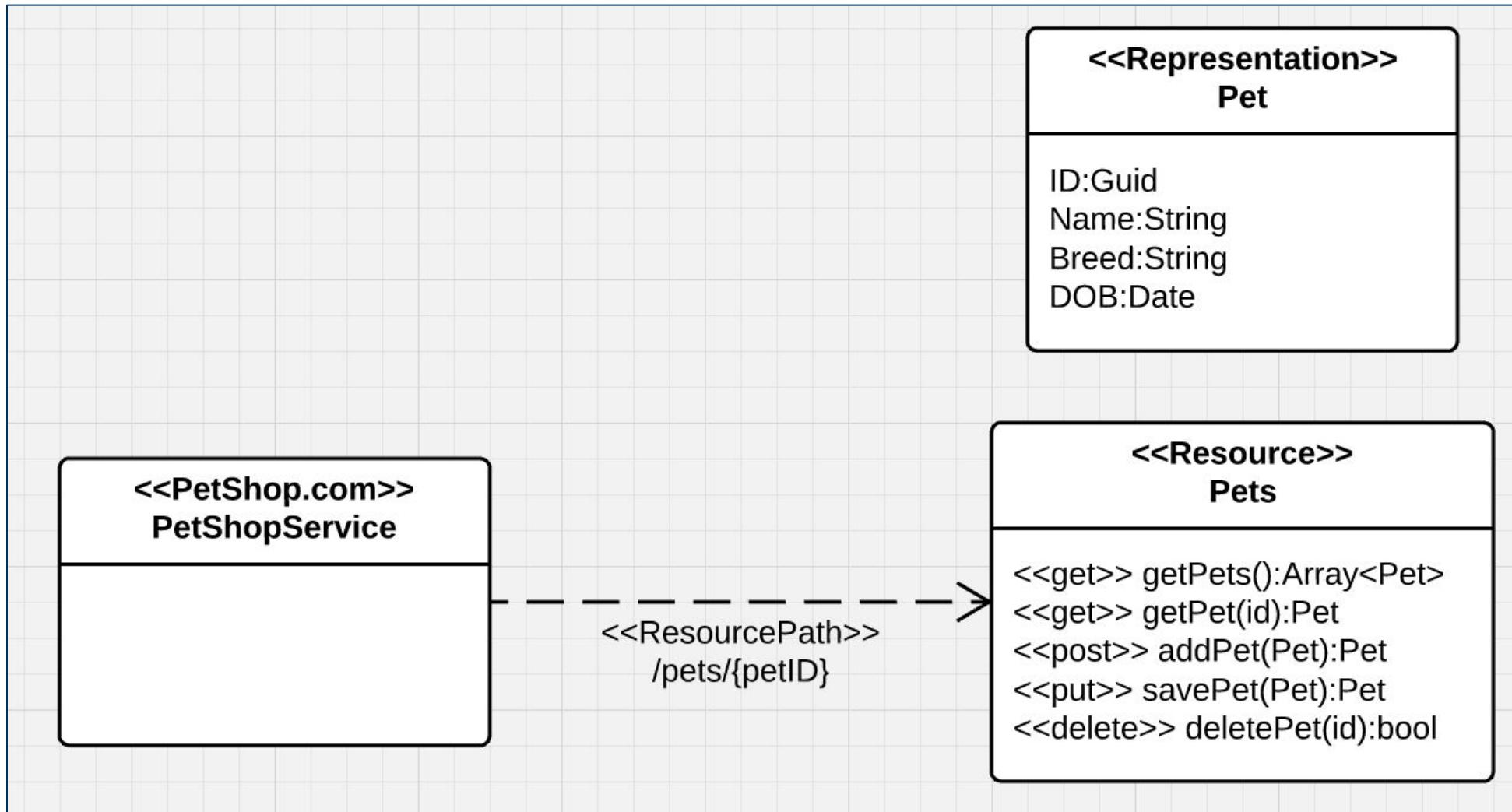
# Recommendations for URI

- Plural nouns for sets (collections)
  - getPets would return a list of pets
- Singular nouns for individual resources
  - getPet would return one pet
- Strive for consistent naming
- URI is case-insensitive
- Don't use verbs to identify a resource
- Include version information

# HTTP Verbs

- **GET** is used to retrieve data
- **POST** is used to add data
- **PUT** is used to add data or alter existing data
  - *Put should be idempotent which means whether 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>

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

# 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/>

# HTML

- HTML is the language of web pages
  - Used to display pages and to send data to REST endpoints
- A basic HTML document contains one head and one body element

```
<!doctype>
<html>
<head>
<title>A Basic Web Page</title>
</head>
<body>
<h3>Font types</h3> <!-- comment won't show up in browser view -->
<p><b>Bold</b>, <i>Italics</i>, <tt>Type-writer</tt> are all
<em>easy</em>
but do not go overboard, because they can get hard to read.</p>
</body>
</html>
```

# HTML Forms

- Allow web pages to submit content to the server
  - Either to server frameworks/scripts or to REST endpoints via AJAX
- Outer `<form>` element has `action` and `method` attributes
- Nested `<input />` elements allow data entry and submission

```
<form action="/login" method="post">
    username: <input name="username" type="text" /> <br />
    password: <input name="password" type="password" /> <br />
    <input type="submit" name="submit" value="submit" />
</form>
```

# 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)

# Chapter Summary

In this chapter, you have:

- Designed systems using a microservice style architecture
- Followed 12-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 12-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



# Application Lifecycle Management

# Chapter Objectives

In this chapter, you will:

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

# Agenda

## 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
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`**

# Agenda

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 master	Push local commits to the remote master repository.
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 navigation 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.' with 'Edit' and 'Add topics' buttons. Below, it shows 9 commits, 1 branch, 0 releases, and 1 contributor. The commits list includes:

File / Action	Description	Time Ago
.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

The screenshot shows a software application interface for managing source code. On the left is a sidebar with icons for Development (a hexagon), Source Code (a blue icon with arrows), Repositories (a gear), and Tools and plugins (a wrench). The 'Source Code' item is selected and highlighted in blue. The main area is titled 'Source Code' and displays a search interface with dropdown menus for 'converter' and 'master'. Below this is a navigation bar with a '/' icon and a dropdown arrow. The main content area lists files with their latest commit details:

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 interface. On the left, there is a sidebar with navigation links: Dashboard, Code (selected), Commits, Commit Visualizer, Triggers, and Settings. The main area is titled "Code: converter". It displays a list of files and folders under the "converter" branch. The files listed are: .idea, templates, application.py, buildspec.yml, Converter.py, converterTests.py, and requirements.txt. There are dropdown menus for "Branch: master" and "Clone URL". A "Connect" button is also visible.

# 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

# Agenda

Package Management

---

Source Control

---

**Version Control**

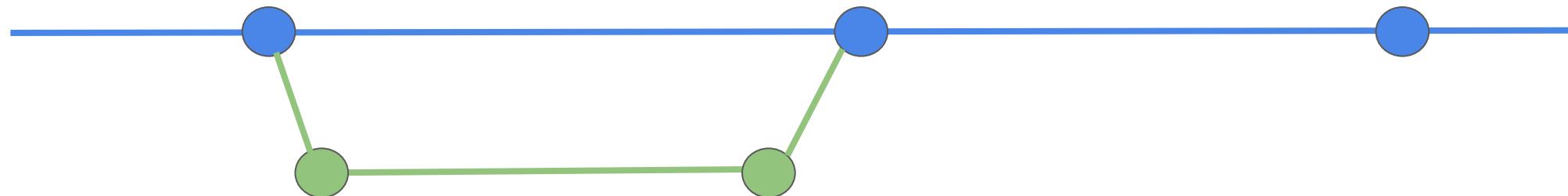
---

# 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 master branch
  - Add branches to make code changes without affecting the master
  - Branches can later be merged to incorporate changes into the master
  - Or branches can be discarded

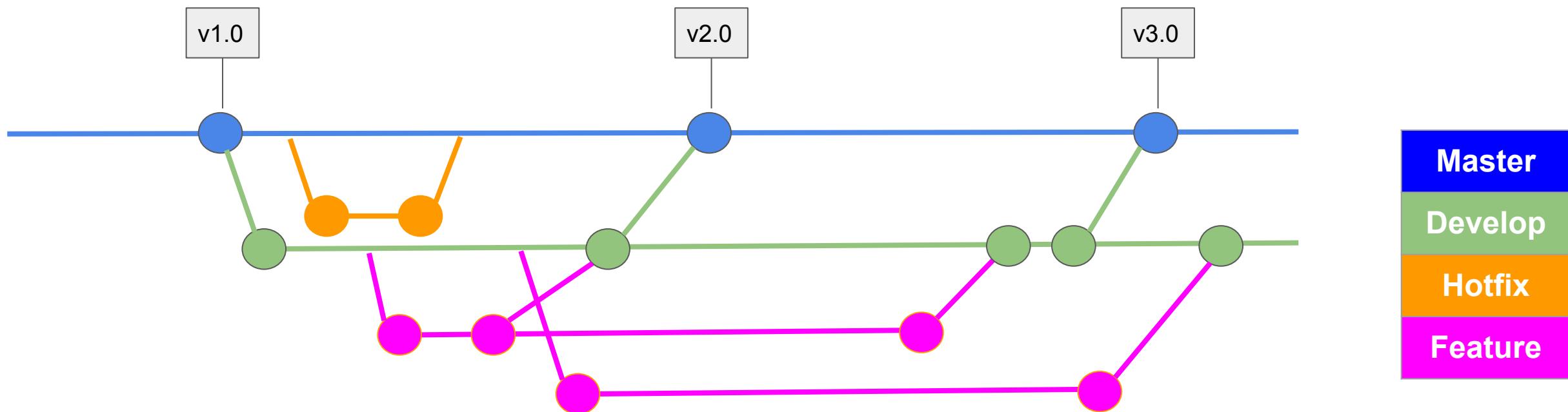
# Branches

- 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 master:
  - Switch to the master: `git checkout master`
  - Merge code changes: `git merge my-branch`



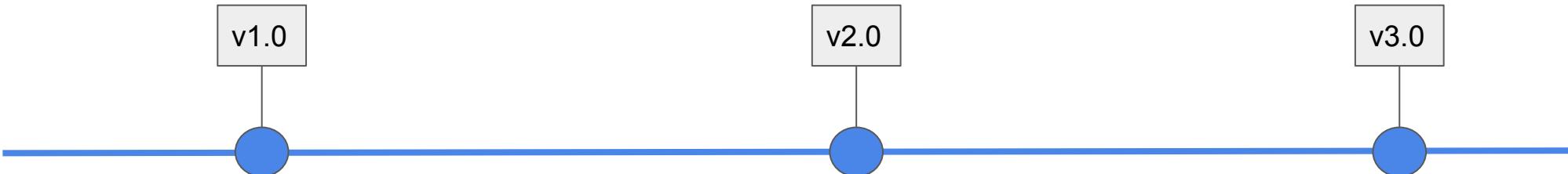
# Branching and Merging Strategies

- The master 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 master
- 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:

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

# 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. What tool below 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

# Agenda

## Understanding Docker

---

Using Docker

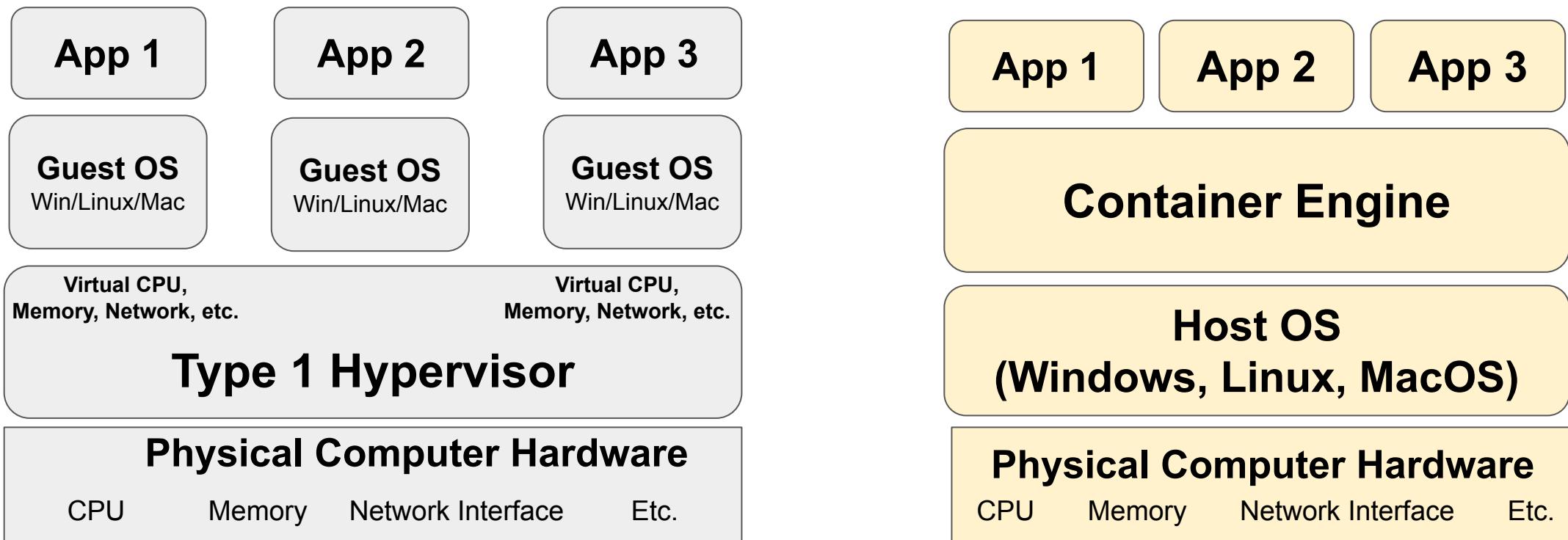
---

Deploying Docker Containers

---

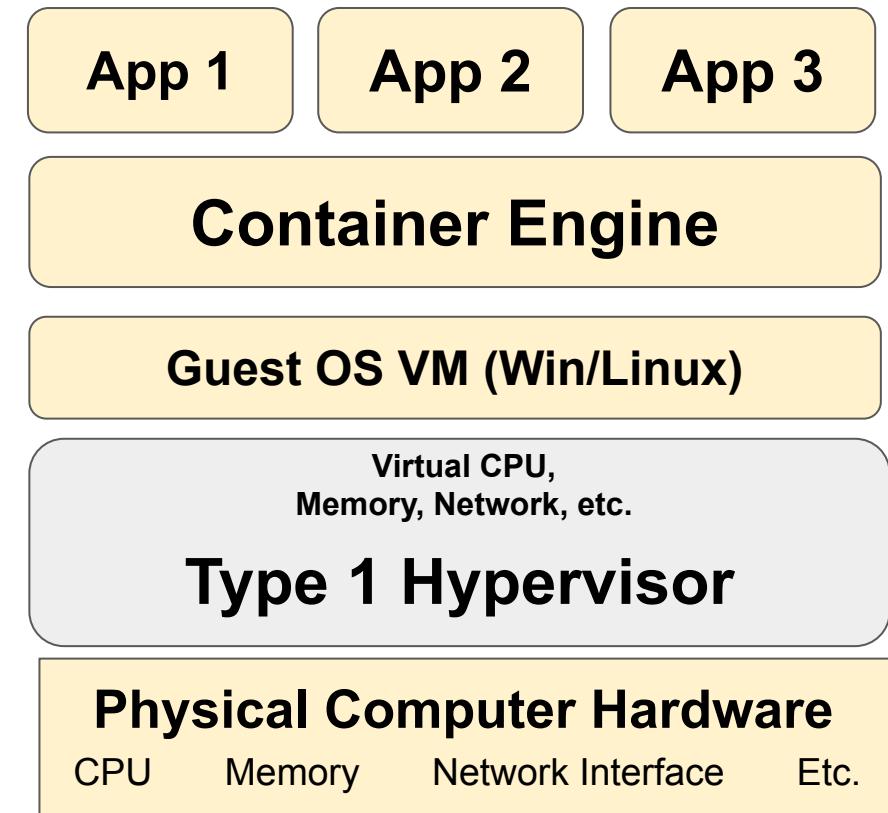
# Containers

- 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

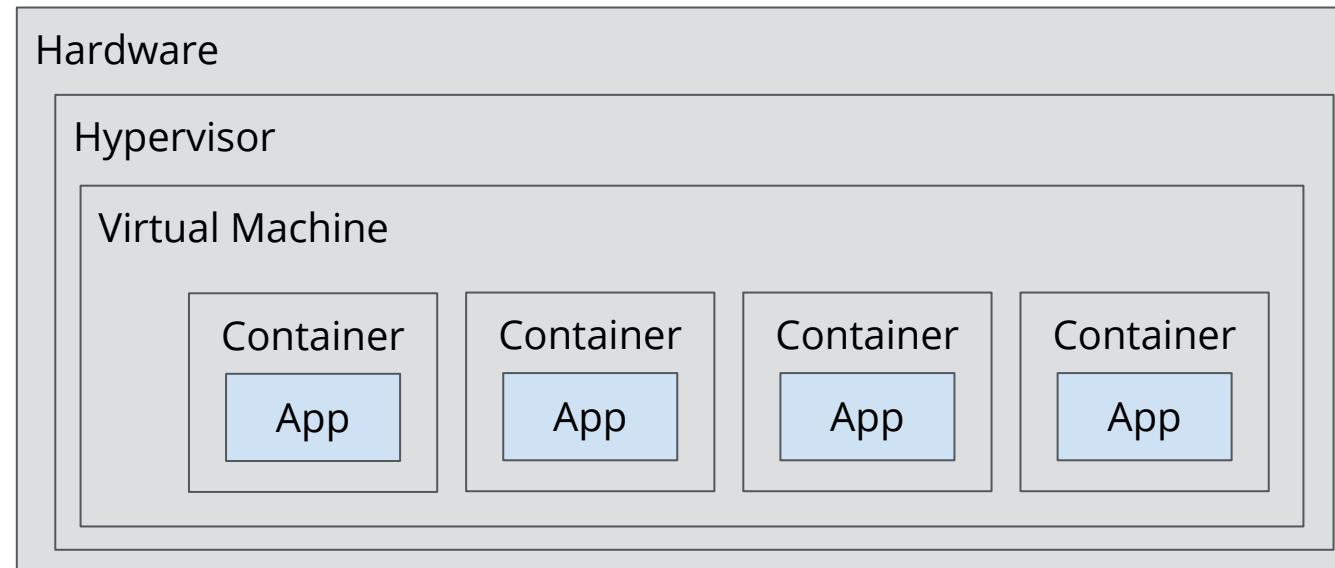
# Images

- 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

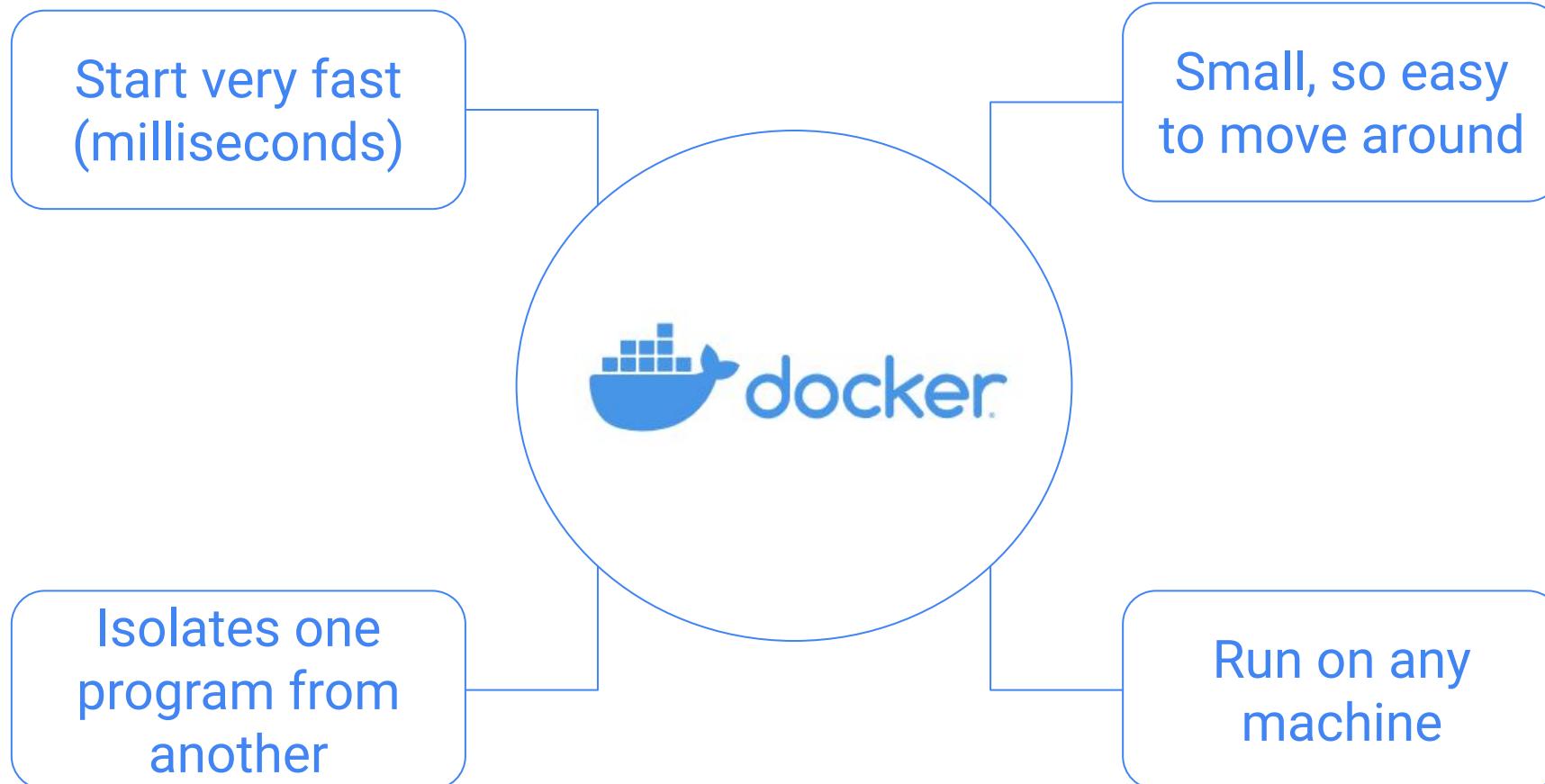


# Containers

- 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



# Agenda

Understanding Docker

---

**Using Docker**

---

Deploying Docker Containers

---

# Getting Started with Docker

## Try this for homework:

- Visit <https://www.docker.com/> and create a Docker ID
  - Docker ID allows you to use the Docker Hub Repository
- Install Docker on your computer
  - Allows Docker commands to be executed
  - Allows Docker containers to run on your machine
- Learn some basic Docker commands
- Learn to build a Docker container for your preferred language
- Upload your container to a repository
- Run your container on a server

# Some Basic Docker Commands

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

# Creating Custom Docker Containers

- 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. 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 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 a Node.js App

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

# 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 .`

# Example Build Command Output

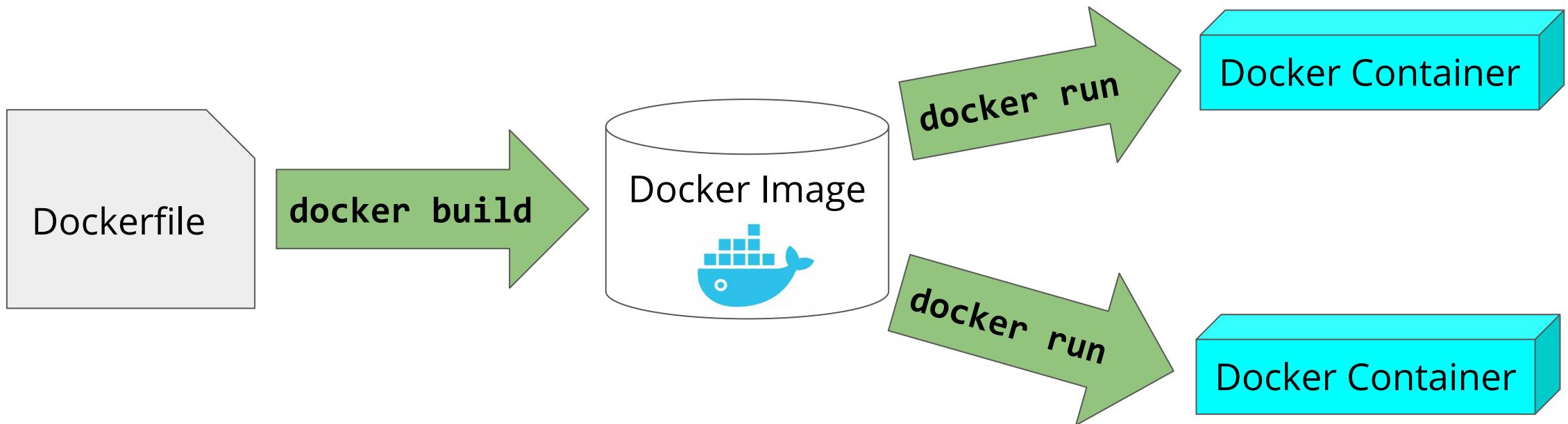
```
$ docker build -t drehnstrom/devops-demo:v0.1 .
Sending build context to Docker daemon 2.828MB
Step 1/7 : FROM python:3.7
--> 34a518642c76
Step 2/7 : WORKDIR /app
<< CODE OMITTED>>
Step 6/7 : ENV PORT=8080
--> Using cache
--> 7045daaaaf44Step 7/7 : CMD exec gunicorn --bind :$PORT --workers 1 --threads
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

# Agenda

Understanding Docker

---

Using Docker

---

**Deploying Docker Containers**

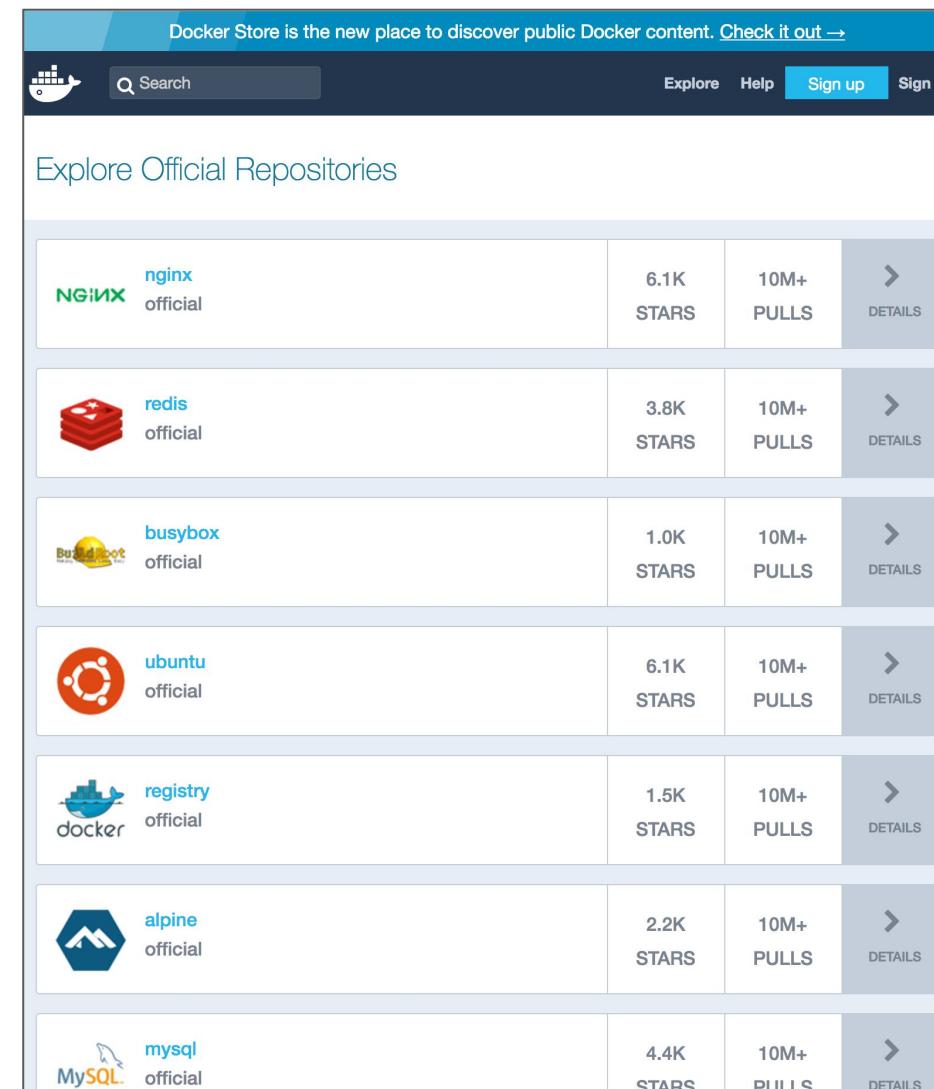
---

# 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

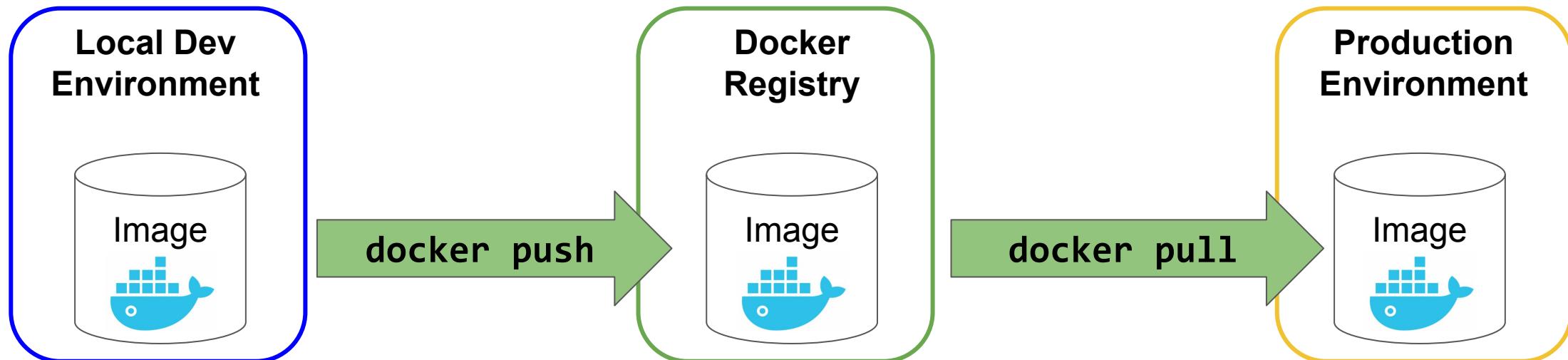
# Docker Hub

- 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)				<a href="#">View push commands</a>	<a href="#">Delete</a>	<a href="#" style="background-color: orange; color: white;">Create repository</a>
<input type="text" value="Find Repositories"/> 				 1 		
Repository name	URI		Created at			
<a href="#">devops-demo</a>	 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 .
```

# 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
- For this activity, we will use Google Container Registry
  - But another registry such as Docker Hub would work as well

# 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

# Agenda

## Kubernetes Clusters

---

Kubernetes

---

Kubernetes Security

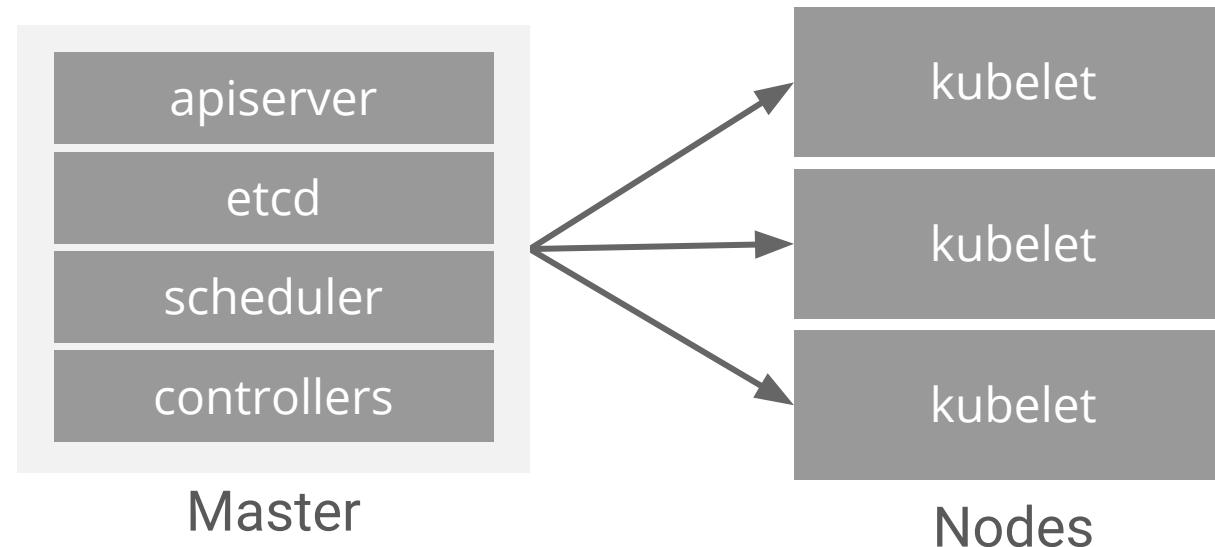
---

# Kubernetes

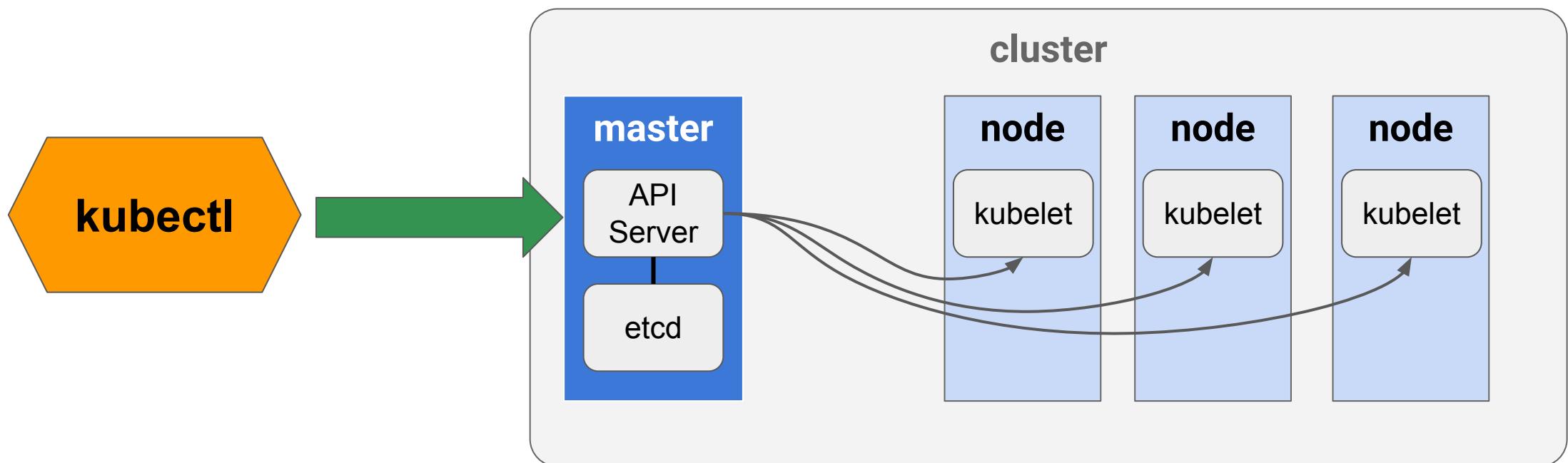
- Open-source container orchestration system
  - Extremely popular and active open-source project
- Originally developed by Google to run Google's data centers
  - Designed to operate at Google scale
  - Proven and tested running Google's applications
- Wide support
  - Used on Google Cloud when using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

# Kubernetes Architecture

- Kubernetes applications require a cluster of machines to run on
  - One or more machines are designated as the master
  - Multiple machines are added to the cluster as nodes
- The master 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 master:

```
$ 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

# AWS Elastic Kubernetes Service (EKS)

EKS > Clusters

Clusters (1)



Delete

Create cluster

Find clusters by name

< 1 >

Cluster name

prod

```
$ 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 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 contains a navigation tree for EKS, with 'Getting Started with eksctl' selected. The main content area is titled 'Getting Started with eksctl' and describes the guide's purpose: to help install required resources for using eksctl to manage Kubernetes clusters. It includes sections on prerequisites, installing the AWS CLI, and instructions for using pip and Python. The top navigation bar includes links for GitHub, PDF, and RSS, language selection (English), and a 'Sign In to the Console' button.

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

'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

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

# 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

SEND FEEDBACK

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

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 [?](#)  
└─ \* Resource group [?](#)  
    Pay-As-You-Go  
    (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 has a 'Quickstarts' section with the following items:

- > Overview
- ▽ Quickstarts
  - ▽ Create an AKS Cluster
    - Use the Azure CLI
    - Use the Azure portal
    - Use a Resource Manager template
  - ▽ Develop applications
    - Use Draft
    - Use Java (VS Code & CLI)
    - Use .NET Core (VS Code & CLI)
    - Use .NET Core (Visual Studio 2017)

# OpenShift

- Manages Kubernetes that runs on Red Hat OpenStack
  - Useful for private cloud deployments
  - Online, managed version is available

The screenshot shows the official Red Hat OpenShift website. At the top, there's a black header bar with the Red Hat logo and "Red Hat OpenShift" text on the left, and "PRODUCTS" and "LEARN" dropdown menus on the right. Below the header, a large red banner features the text "RED HAT OPENSHIFT 4 IS HERE" in red, followed by "The Kubernetes platform for big ideas" in black. A subtext below it reads "Empower developers to innovate and ship faster with the leading hybrid cloud, enterprise container platform". At the bottom of the banner are two buttons: a red "Get started" button and a white "What's new in OpenShift 4?" button.

# 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

- Create a Kubernetes cluster in the cloud
  - Once the cluster is created, using the cluster is exactly the same no matter where the cluster is running (AWS, Azure, etc.)

# Agenda

Kubernetes Clusters

---

**Kubernetes**

---

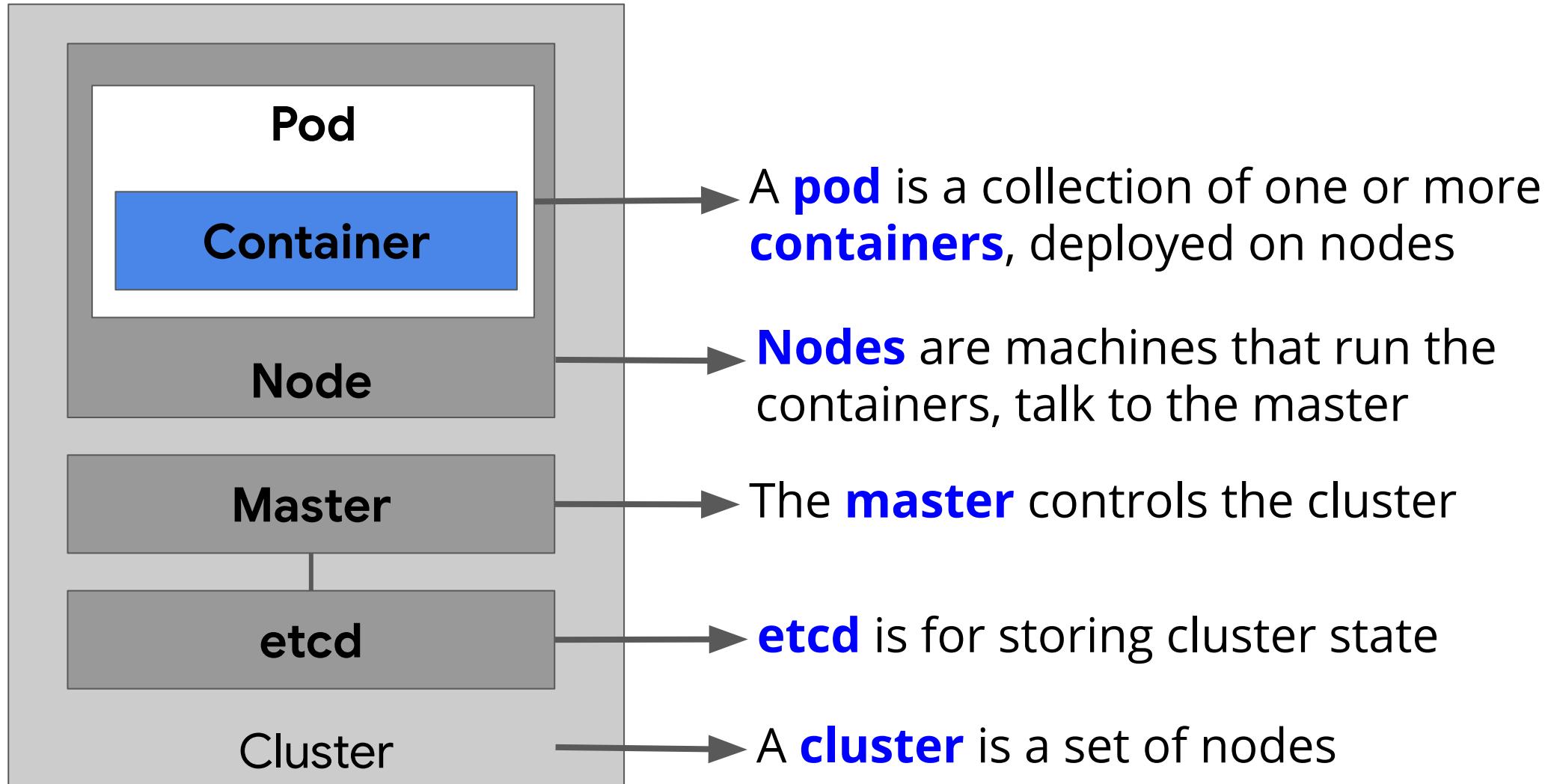
Kubernetes Security

---

# 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
- 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
- Deployments are configurations that define service resources

# 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
- Foundation
  - See: <https://kubernetes.io/>

kubectl Command	Description
kubectl get nodes	Show information about the cluster nodes.
kubectl create deployment [name] [docker-image] [replicas] kubectl create deployment spaceinvaders --image=drehnstrom/space-invaders --replicas=3	Start one or more instances of a Docker image (pods). This is a deployment.
kubectl get pods	Show the running pods.
kubectl scale [deployment name] [replicas] kubectl scale deployments/spaceinvaders --replicas=6	Change the number of pods in a deployment.
kubectl expose [deployment name] [name] [port] [type] kubectl expose deployments/spaceinvaders --name=space-lb --port=80 --type=LoadBalancer	Create a service that provides access to the deployment (the pods).
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).

# Starting Pods with kubectl

Runs 3 instances of the specified Docker image in the cluster

```
$ kubectl create deployment spaceinvaders --image=drehnstrom/space-invaders  
--replicas=3
```

```
deployment.apps/spaceinvaders created
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
spaceinvaders-55c88695bb-7n9cd	1/1	Running	0	12s
spaceinvaders-55c88695bb-c4p8s	1/1	Running	0	12s
spaceinvaders-55c88695bb-qdqpc	1/1	Running	0	12s

# Use kubectl Scale to Add Instances

Scale the deployment  
to 6 instances

```
$ kubectl scale deployments/spaceinvaders --replicas=6
```

```
deployment.extensions "spaceinvaders" scaled
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
spaceinvaders-55c88695bb-7n9cd	1/1	Running	0	59s
spaceinvaders-55c88695bb-c4p8s	1/1	Running	0	59s
spaceinvaders-55c88695bb-hkhbp	0/1	ContainerCreating	0	4s
spaceinvaders-55c88695bb-ktkt4	1/1	Running	0	4s
spaceinvaders-55c88695bb-1jmvm	1/1	Running	0	4s
spaceinvaders-55c88695bb-qdqpc	1/1	Running	0	59s

# Use kubectl Expose to Allow Containers to Be Accessed

Create a load balancer so the pods are reachable

```
$ kubectl expose deployments/spaceinvaders --name=space-lb --port=80  
--type=LoadBalancer
```

```
service "space-lb" exposed
```

```
$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.11.240.1	<none>	443/TCP	24m
space-lb	LoadBalancer	10.11.243.98	35.188.2.190	80:30144/TCP	2m

A load balancer has an external IP address

# 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
$
```

# Use kubectl delete to Get Rid of What You Created

```
$ kubectl delete services/space-lb
service "space-lb" deleted
$ kubectl delete deployments/spaceinvaders
$ kubectl get pods
No resources found.
$ kubectl get services
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP     10.11.240.1   <none>        443/TCP       38m
```

# Pods Can Be Defined in Configuration Files

```
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

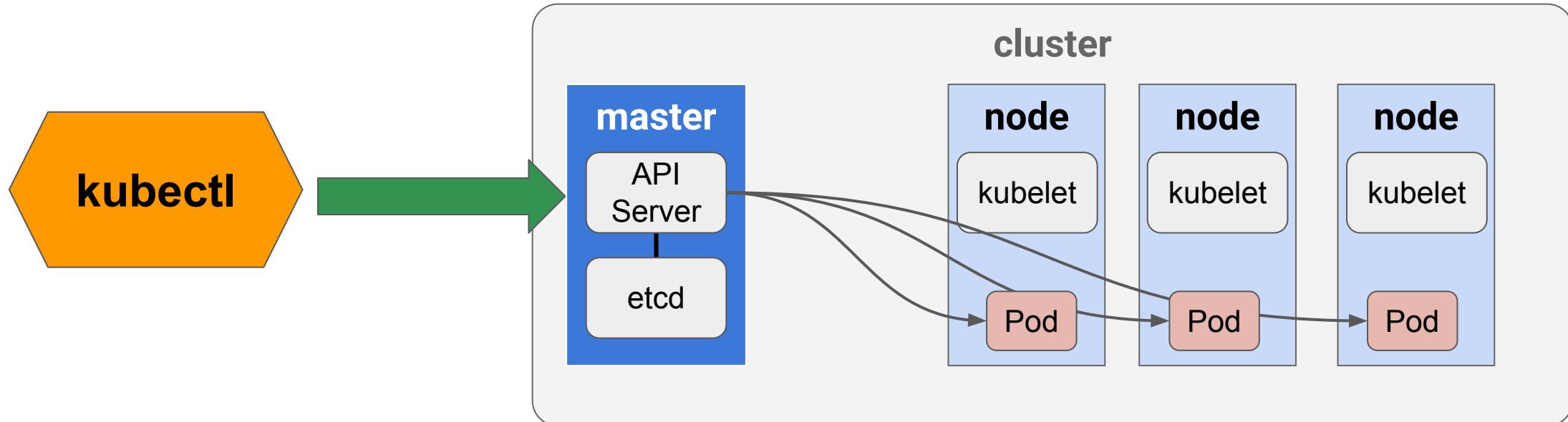
We want 3 replicas of the pod

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

# Kubernetes Deployments

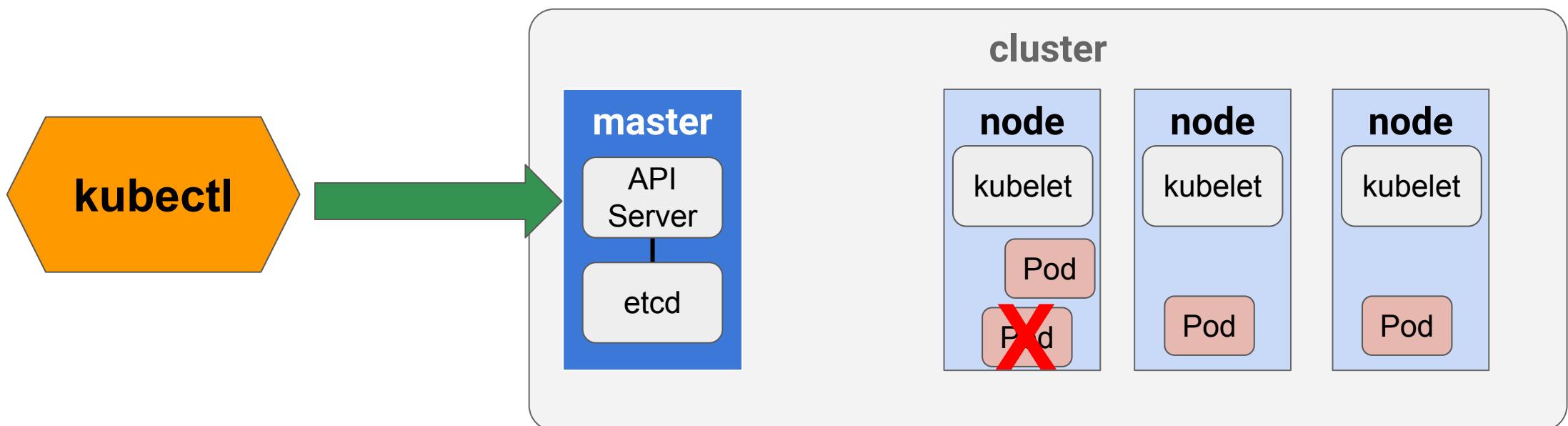
- Use kubectl to send the config file to the master
  - The master then decides how to deploy the pods

```
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 a service 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
```

# Accessing a Deployment with a Load Balancer

- Need a load balancer to route requests to the pods

```
kubectl expose deployment devops-deployment --port=80  
--target-port=8080 --type=LoadBalancer
```

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

```
kubectl get services
```

# Adding the Load Balancer to Configuration

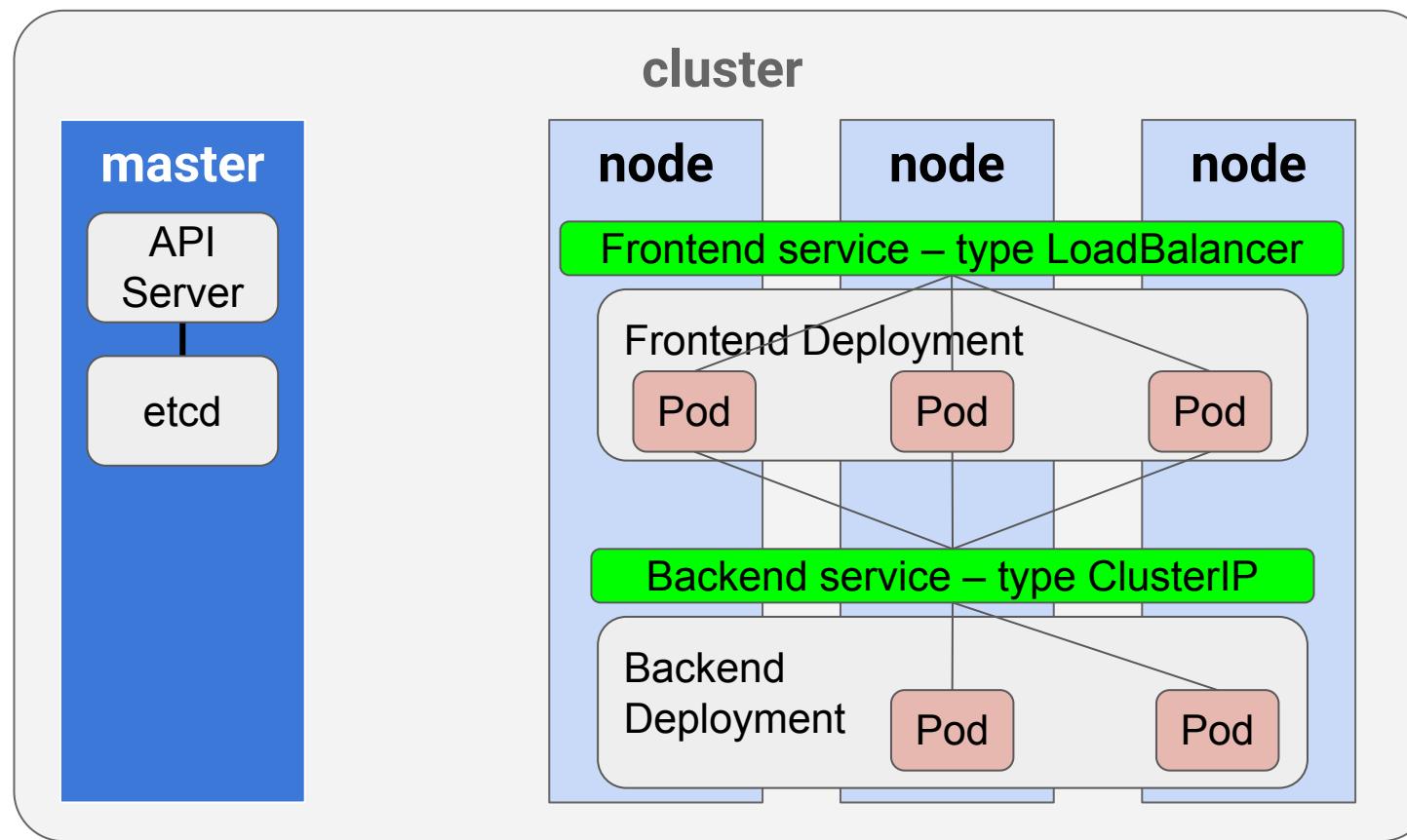
```
apiVersion: v1
kind: Service
metadata:
  name: devops-loadbalancer
  labels:
    app: devops
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: devops
    tier: frontend
```

# 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

- To dynamically scale up and down, create an autoscaler
  - Specify min and max number of pods and some metric to monitor

```
kubectl autoscale deployment devops-deployment --min=5  
--max=10 --cpu-percent=60
```

# Adding the Autoscaler to 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
- Can also delete resources individually when created at the command line

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

```
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

# Tutorials: Using Kubernetes

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](#)

# 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)

# Agenda

Kubernetes Clusters

---

Kubernetes

---

**Kubernetes Security**

---

# Securing a Kubernetes Cluster

- Use a minimal OS
- Use RBAC
- Use namespaces
- 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: [""]
  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

# Use Namespaces

- Namespaces allow a single Kubernetes cluster to be divided into multiple “virtual clusters”
  - Can be used to divide cluster resources between multiple users
  - Multiple namespaces inside a single Kubernetes cluster are logically isolated from each other
  - Can help with organization, security, and even performance
- By default, Kubernetes clusters will have a namespace called **default**
  - Actually three namespaces (default, kube-system, kube-public)
  - **kube-public** could be used to share configmaps across namespaces
  - **kube-system** should be left alone

# Creating Namespaces

```
kind: Namespace  
apiVersion: v1  
metadata:  
  name: development  
  labels:  
    name: development
```

Configuration to create a namespace

```
$ kubectl apply -f pod.yaml --namespace=development
```

Use the namespace parameter to put resources in a particular 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: MWYyZDFlMmU2N2Rm
```

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 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

# 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

# Agenda

## **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
  - 99.9999999%
- 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

# On-Premises vs. Cloud Storage

On-Premises Storage	Cloud Storage
Space is limited and expensive	Space is nearly infinite and cheap
Data collection can be slow	Data can be collected at great speed
Data retrieval can be slow	Gigabytes and even terabytes can be read in seconds
Complex administration required	Managed services available
Security must be managed	Security built in

# Amazon S3 Storage 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 S3 Storage 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
- 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

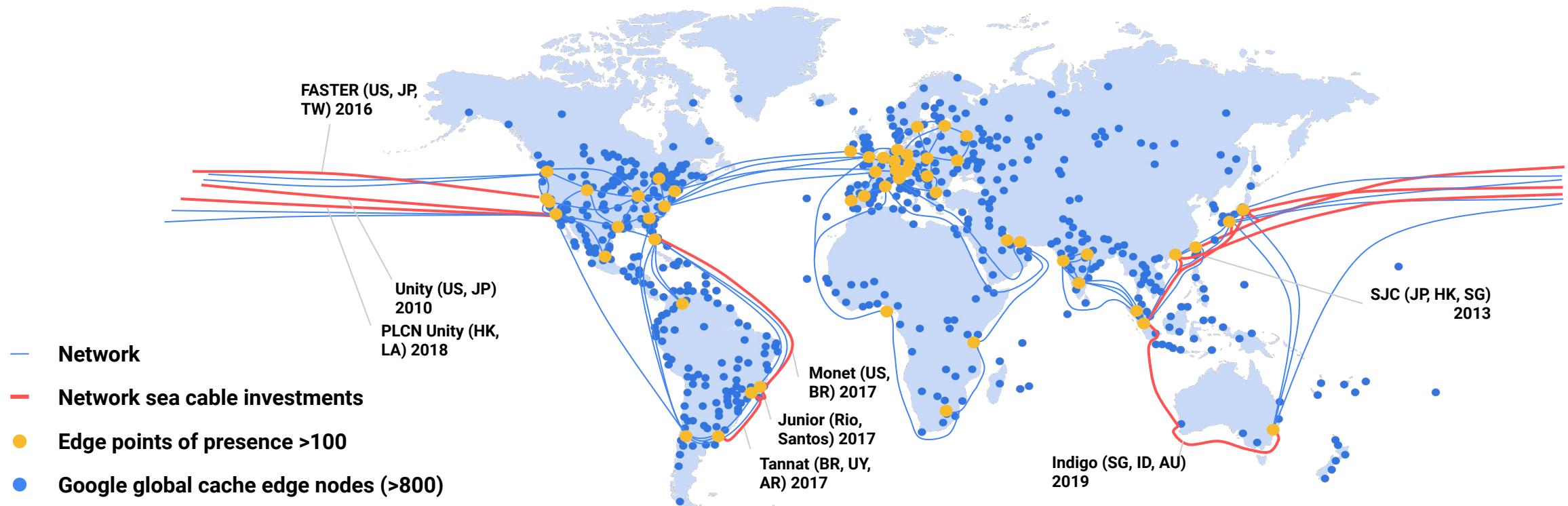
- Available in three replication classes
  - Locally Redundant Storage (LRS)
    - Replicated synchronously to three different storage nodes within the same region
  - Geo Redundant Storage (GRS)
    - Default option
    - Same as LRS plus asynchronous replication to another region
  - Read-Access Geo 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



# Tutorials: Cloud-Based Binary Storage

- AWS S3
  - <https://www.qwiklabs.com/focuses/16438?parent=catalog>
- Google Cloud Storage
  - <https://www.qwiklabs.com/focuses/569?parent=catalog>

# Agenda

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
  - Spanner is a horizontally, global-scalable, and strongly consistent database
- Amazon
  - 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

# Spanner and Financial Services

- A few good articles on Spanner

- Nick Weisfeld

<https://www.finextra.com/blogposting/14218/cloud-database-technology-can-no-longer-be-ignored-in-financial-services>

- Andrew Rossiter

[https://www.finextra.com/blogposting/14465/google-cloud-spanner-the-next-big-transformation-in-financial-services?utm\\_medium=dailynewsletter&utm\\_source=2018-9-6](https://www.finextra.com/blogposting/14465/google-cloud-spanner-the-next-big-transformation-in-financial-services?utm_medium=dailynewsletter&utm_source=2018-9-6)

# Tutorials: Managed Relational Databases

- AWS RDS
  - Windows: <https://www.qwiklabs.com/focuses/14430?parent=catalog>
  - Linux: <https://www.qwiklabs.com/focuses/16318?parent=catalog>
- Google Cloud SQL
  - <https://www.qwiklabs.com/focuses/936?parent=catalog>
- Google Spanner
  - <https://www.qwiklabs.com/focuses/1774?parent=catalog>

# Agenda

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
  - Firestore/Datastore
  - Bigtable
- AWS
  - SimpleDB
  - DynamoDB
- 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
  - 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

- SimpleDB
  - Key-Value store
  - Highly available, simple, secure
  - Inexpensive with a free tier of 1GB per month
  - <https://aws.amazon.com/simpledb/>
- DynamoDB
  - Supports both Document and Key-Value store models
  - Very fast and scalable
  - <https://aws.amazon.com/dynamodb/>

# Tutorials: NoSQL Data Services

- AWS
  - <https://www.qwiklabs.com/focuses/14815?parent=catalog>
- Google
  - <https://www.qwiklabs.com/focuses/941?parent=catalog>

# 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

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

# Agenda

Storing Binary Data

---

Relational Data Service

---

NoSQL Data Services

---

**Caching**

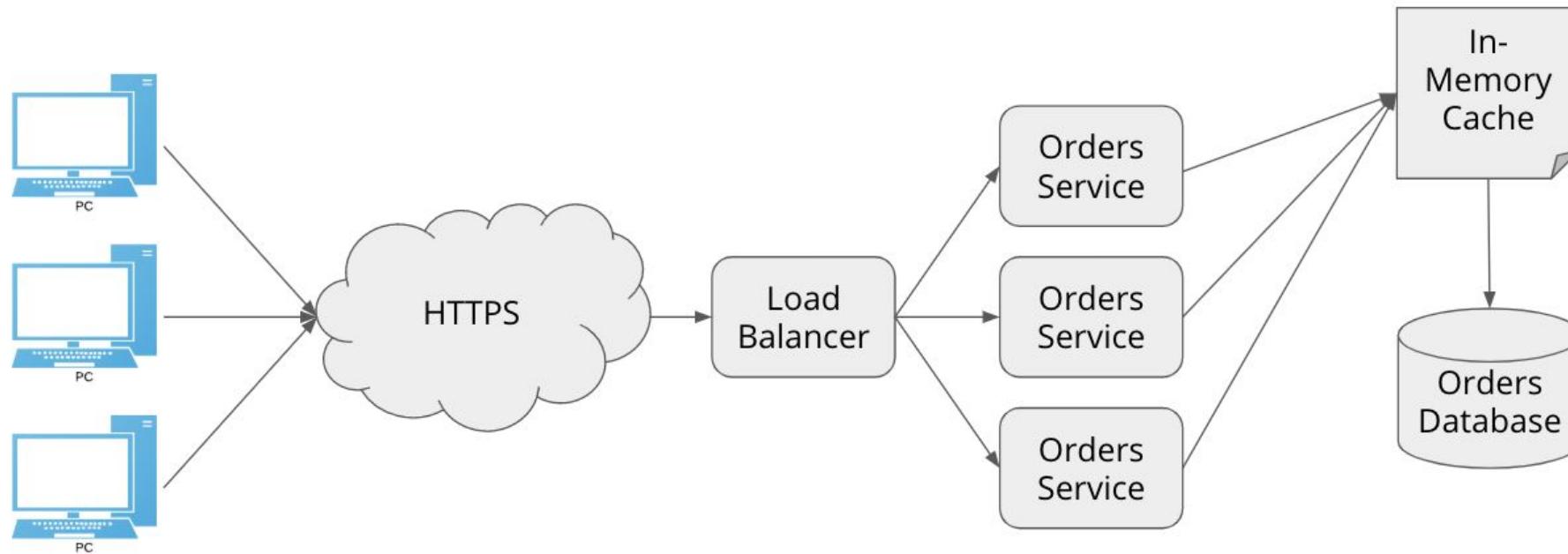
---

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

- AWS ElastiCache
  - Managed, Redis or Memcached-compatible in-memory data store
- Google Memorystore
  - Fully-managed in-memory data store service for Redis
- Google App Engine Memcache
  - Offers a free, shared service level

# Activity: Using Data Services

- Review your case study design and decide which data services may be useful to achieve the requirements
  - Storage
  - Relational Database
  - NoSQL Database
  - Caching
- Modify one or more of your microservices to use a data service

# Agenda

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

# AWS 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

# AWS 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 store
- B. Relational database
- C. NoSQL database
- D. Data Warehouse

# Quiz

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

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

# Quiz

**You want to gather data from different sources like 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 store
- 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

# Agenda

## Automated Platforms

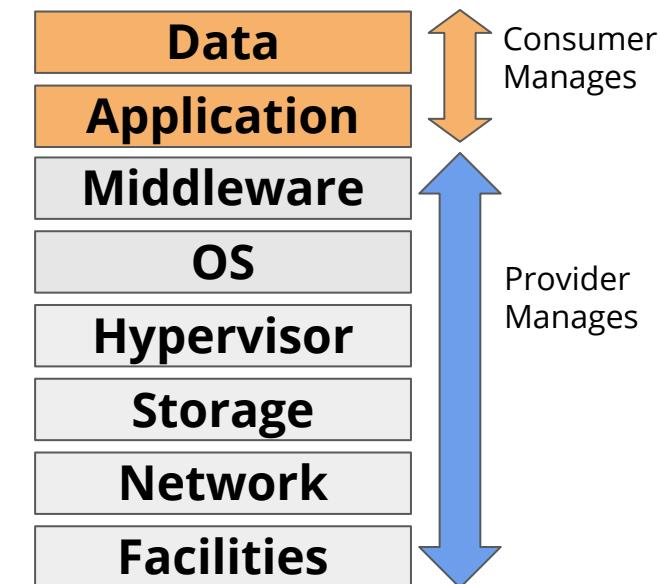
---

### 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
  - <https://www.qwiklabs.com/focuses/3340?parent=catalog>
- AWS Elastic Beanstalk
  - <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html>

# Agenda

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 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
- 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 severals ways
  - An HTTP call
  - In response to a file added to a storage bucket
  - When a message is posted to a message queue
  - Etc.

# 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

# 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
  - <https://www.qwiklabs.com/focuses/15682?parent=catalog>
- Google Cloud Functions
  - <https://www.qwiklabs.com/focuses/1763?parent=catalog>
- Google Cloud Run
  - <https://www.qwiklabs.com/focuses/5162?parent=catalog>

# 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 like 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 what 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
- Use tools for analyzing code quality and testing
- 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

# Agenda

## Automated Builds

---

Code Quality Tools

---

Jenkins

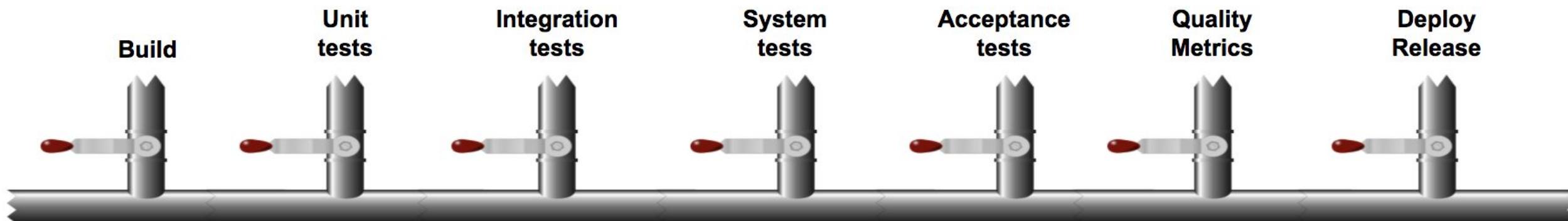
---

# Continuous Integration

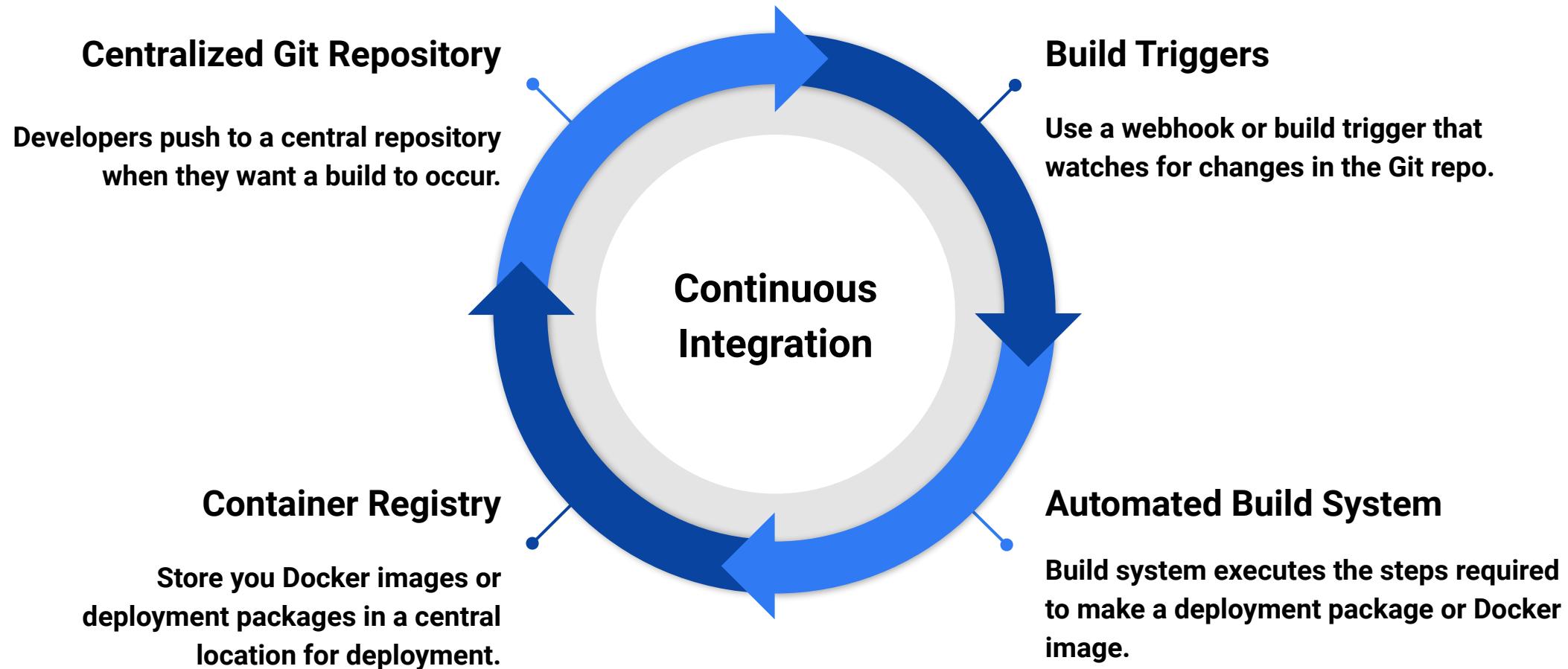
- Continuous integration is ultimately about automation
  - Tools used to make testing, quality, and deployment fast and simple
- Deploying new software or new versions should be no big deal
  - Deployments are done frequently to test environments
  - Deployment to production should be as simple as a 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 master 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' configuration page on GitHub. The left sidebar has a 'Webhooks' section selected. The main form area contains the following fields:

- Payload URL \***: `https://www.mysite.com/build`
- Content type**: `application/x-www-form-urlencoded`
- Secret**: A blank input field.
- SSL Verification**: A note stating "By default, we verify SSL certificates when delivering payloads." with a "Disable SSL verification" button.
- Events**: A section titled "Which events would you like to trigger this webhook?" with three radio button options:
  - Just the push event.
  - Send me everything.
  - Let me select individual events.

# AWS Code Automation

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

# AWS CodeBuild

- 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

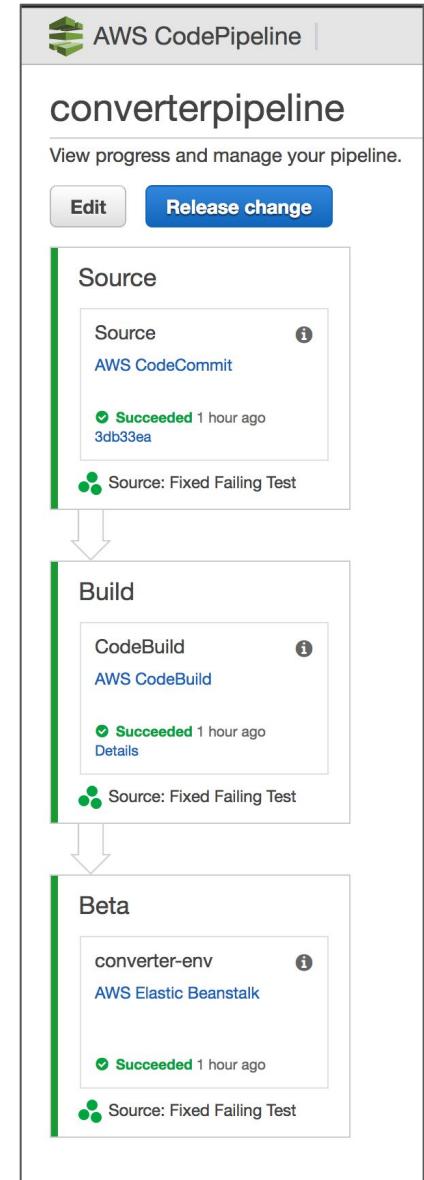
The screenshot shows the AWS CodeBuild console interface. On the left, there's a sidebar with 'AWS CodeBuild' at the top, followed by two menu items: 'Build projects' (which is highlighted with an orange border) and 'Build history'. The main content area has a title 'Build Project: converter'. Below the title are three buttons: 'Back', 'Delete' (in red), and 'Edit project' (in blue). The main section is titled 'Project' and contains the following details:

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

Below these details, 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 Amazon CodeCommit or GitHub repositories
- Can deploy to various deployment platforms including Elastic Beanstalk, Cloud Formation, and others



# Tutorial: AWS Code Build

Do the following if you want to learn about AWS Code Build:

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

# 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

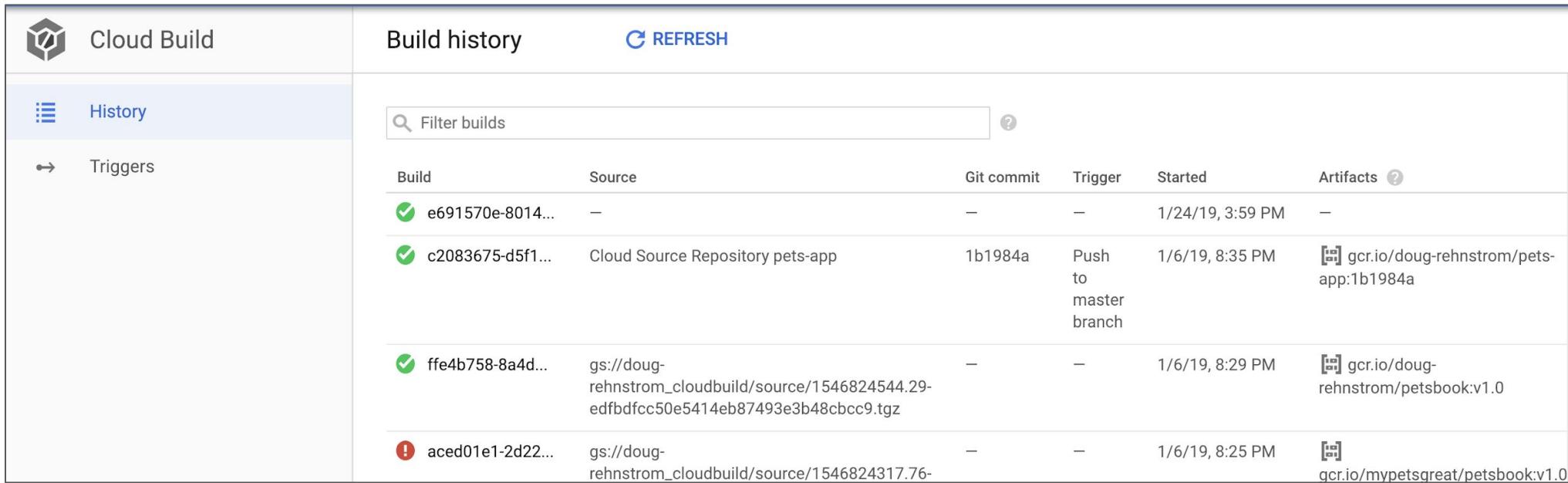
# Cloud Source Repositories

The screenshot shows the Google Cloud Source Repositories interface. At the top, there's a navigation bar with the title "Cloud Source Repositories", a search bar, and various navigation links like "Cloud Console" and "Add repository". Below the navigation bar, the main content area is titled "My Source". It features a section titled "Starred (6)" containing a list of six starred repositories, each with a star icon and the repository name followed by the owner's name: "converter-cloud-function doug-rehnstrom", "devops-converter doug-rehnstrom", "image-index doug-rehnstrom", "pets-app doug-rehnstrom", "space-invaders doug-rehnstrom", and "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	Source	Git commit	Trigger	Started	Artifacts
	✓ 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	 gcr.io/doug-rehnstrom/pets-app:1b1984a
	✓ ffe4b758-8a4d...	gs://doug-rehnstrom_cloudbuild/source/1546824544.29-edfbdfcc50e5414eb87493e3b48cbcc9.tgz	—	—	1/6/19, 8:29 PM	 gcr.io/doug-rehnstrom/petsbook:v1.0
	✗ aced01e1-2d22...	gs://doug-rehnstrom_cloudbuild/source/1546824317.76-	—	—	1/6/19, 8:25 PM	 gcr.io/mypetsgreat/petsbook:v1.0

# Google Cloud Build Triggers

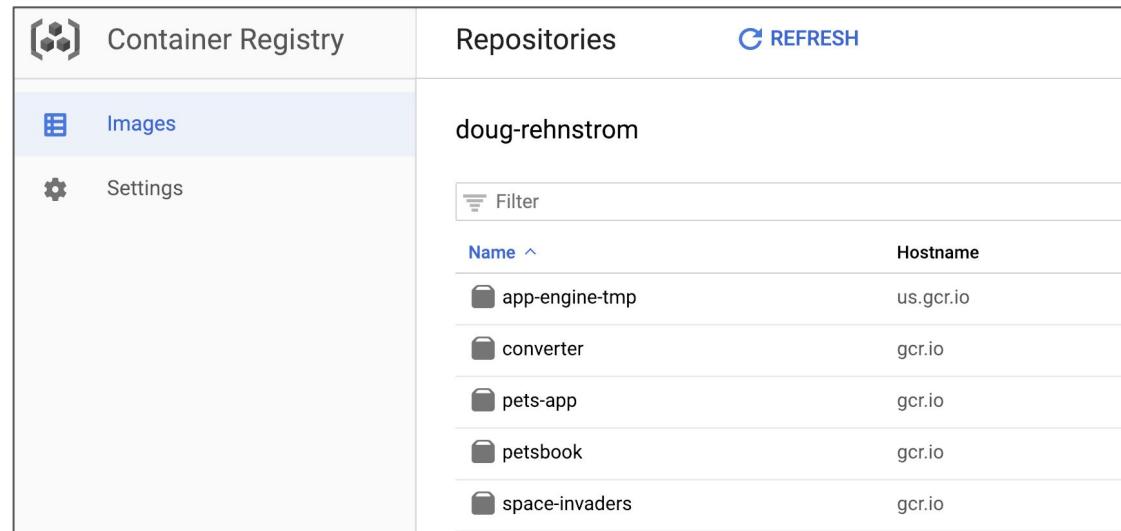
- Watch a repository and build a container whenever code is pushed
  - Works with GitHub and other external git repos
  - Not limited to Docker builds, can use Maven or custom builds

The screenshot shows the Google Cloud Build Trigger creation process across three steps:

- Step 1: Select source**
  - Header: "Create trigger".
  - Sub-header: "Select source".
  - Content: "Choose a repository hosting option".
    - Cloud Source Repository
    - GitHub
    - Bitbucket
  - Buttons: "Continue" (blue) and "Cancel".
- Step 2: Select repository**
  - Header: "Create trigger".
  - Sub-header: "Select repository".
  - Content: "Source: Cloud Source Repository".
    - "Filter repositories" button.
    - default
    - Cloud Source Repository
  - Buttons: "Continue" (blue) and "Cancel".
- Step 3: Trigger settings**
  - Header: "Trigger settings".
  - Content:
    - Source: Cloud Source Repository    Repository: <https://source.developers.google.com/p/remote>
    - Name (Optional): "Build My Docker Container"
    - Trigger type:
      - Branch
      - Tag
    - Branch (regex): "Matches the branch: master" (value: ".")
    - 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" (value: "/")

# 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, there's a sidebar with 'Container Registry' at the top, followed by 'Images' (which is highlighted in blue) and 'Settings'. The main area is titled 'Repositories' and features a 'REFRESH' button. Below that is a 'Filter' section with 'Name ^' and 'Hostname'. A table lists 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:

- <https://cloud.google.com/cloud-build/docs/quickstart-docker>
- <https://cloud.google.com/cloud-build/docs/running-builds/automate-builds>

# Activity: Automated Builds

Create an automated build for your case study project

- You can use GitHub Webhooks, AWS Code Build, Google Cloud Build, or something else

# Agenda

Automated Builds

---

**Code Quality Tools**

---

Jenkins

---

# SonarQube

- Used for static code analysis
  - Includes code quality metrics
  - Detects code smells and common inefficiencies
  - Detects security vulnerabilities
- Works with many different languages
  - Java, JavaScript, C#, TypeScript, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML, and VB.NET
- Can be integrated into a Jenkins CI/CD pipeline
- See: <https://www.sonarqube.org/>

# Tutorial: SonarQube

- <https://docs.sonarqube.org/latest/setup/overview/>

# Agenda

Automated Builds

---

Code Quality Tools

---

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

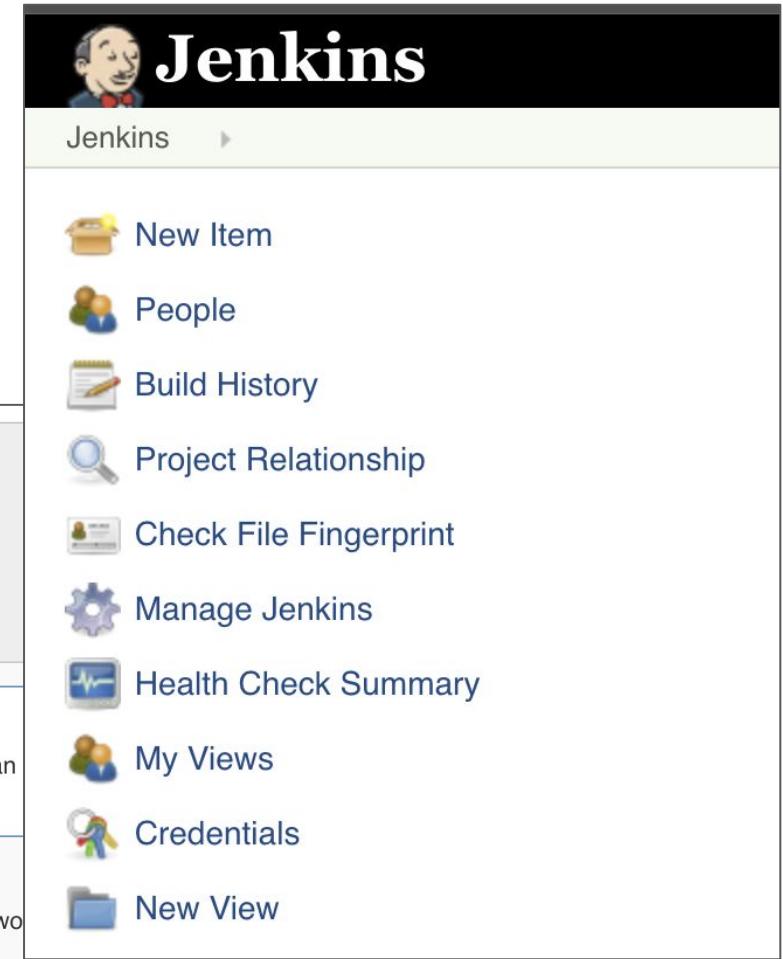
Enter an item name

» 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

Most automated, assuming you are using Maven

Use this for scripted and declarative pipelines (more later)



## 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.



## Maven project

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



## Pipeline

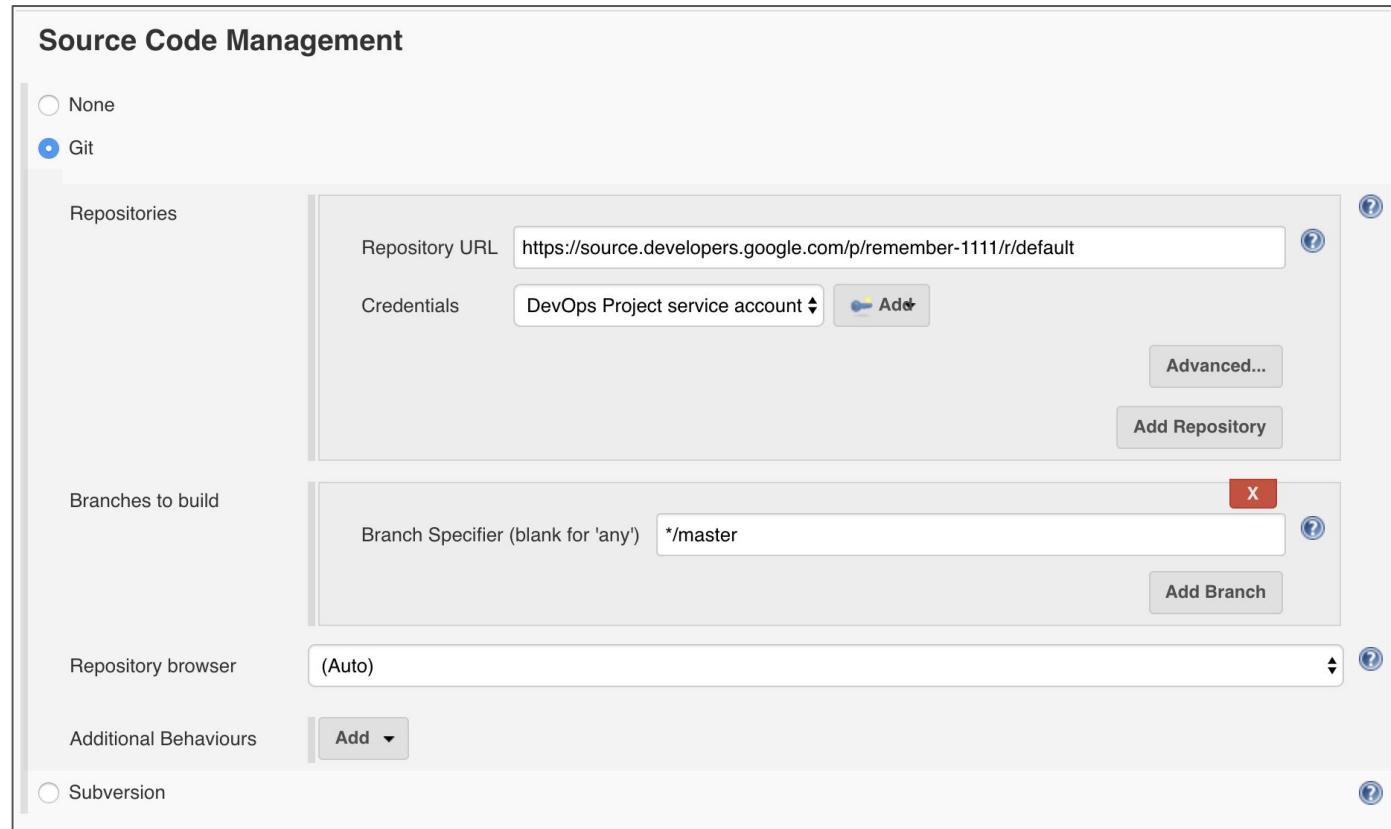
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

# Defining Freestyle Project Pipelines

1. Define where source control is managed
2. Set up build trigger
3. Customize the build environment
4. Respond to trigger with one or more build steps
5. Define post-build steps

# Source Code Management

- Git and Subversion are supported
  - Must include some authentication information to access the 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

Timeout minutes

Time-out variable

Set a build timeout environment variable

Time-out actions

Add timestamps to the Console Output

Google Cloud Ephemeral Deployer

JClouds Single-use slave

Log Build Status to Git Notes

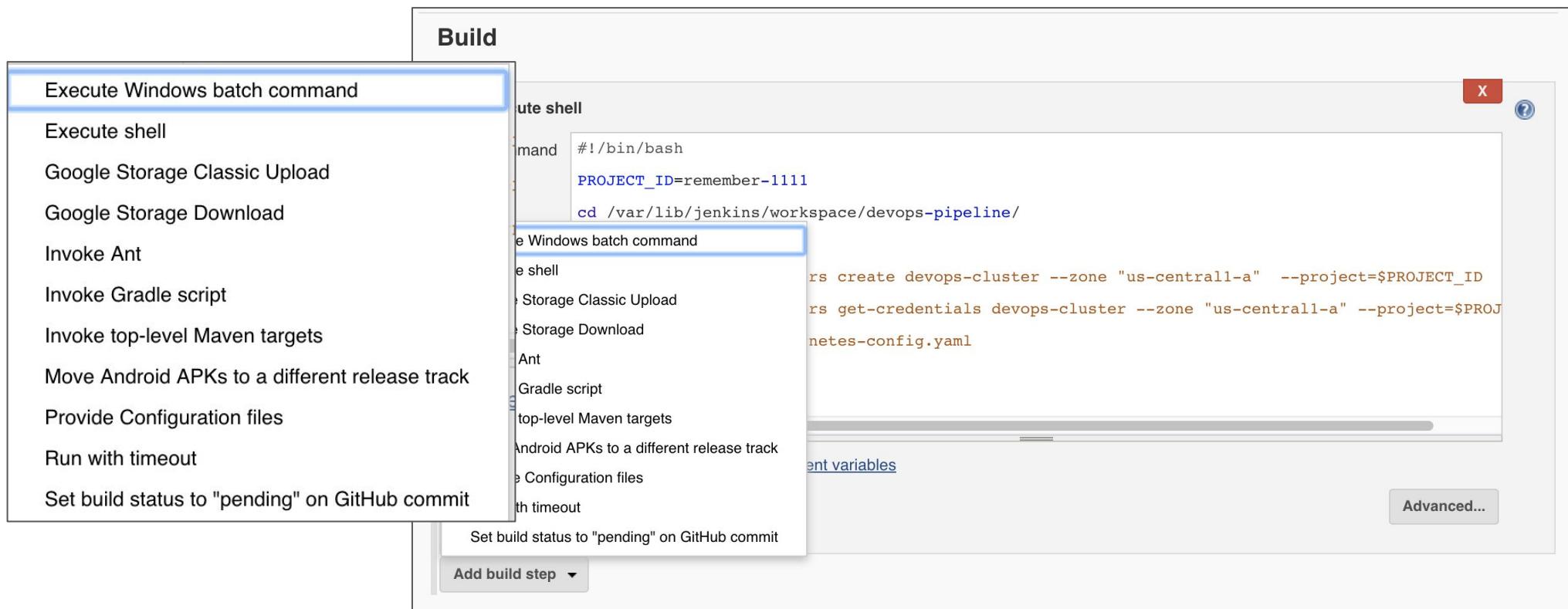
Setup Kubernetes CLI (kubectl)

Use secret text(s) or file(s)

With Ant

# Build Steps

- Can have any number of steps in the pipeline
  - Write a script



# Build Step Script Examples

- Source code is pulled from the repository to the /var/lib/jenkins/workspace/ folder on the Jenkins server
- Script is running from that folder using the Jenkins server account
  - Need to install the software required for the scripts on the server

```
echo 'Install the Python requirements...'
pip3 install -r requirements.txt
```

```
echo 'Run the tests'
python3 -m pytest
```

```
echo 'Create the Docker container'
docker --version
docker build -t gcr.io/doug-rehnstrom/jenkins-converter:v0.2 .
docker images
```

# Post-Build Steps

## Post-build Actions

### E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like 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)

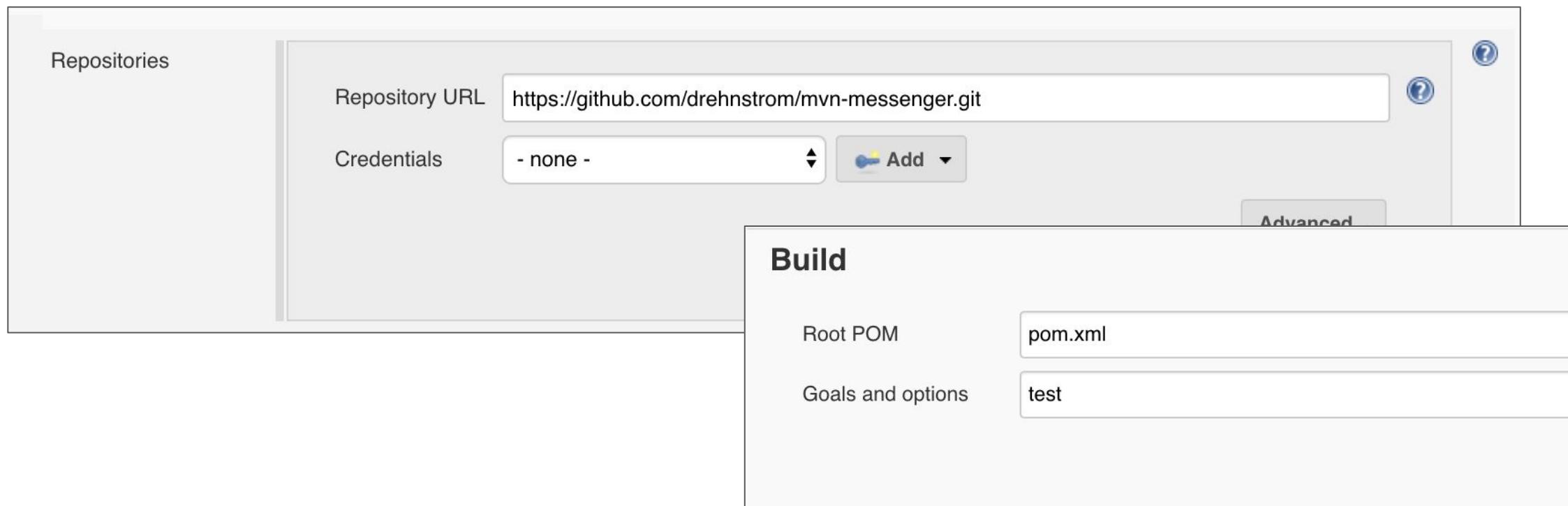
Set build status on GitHub commit [deprecated]

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



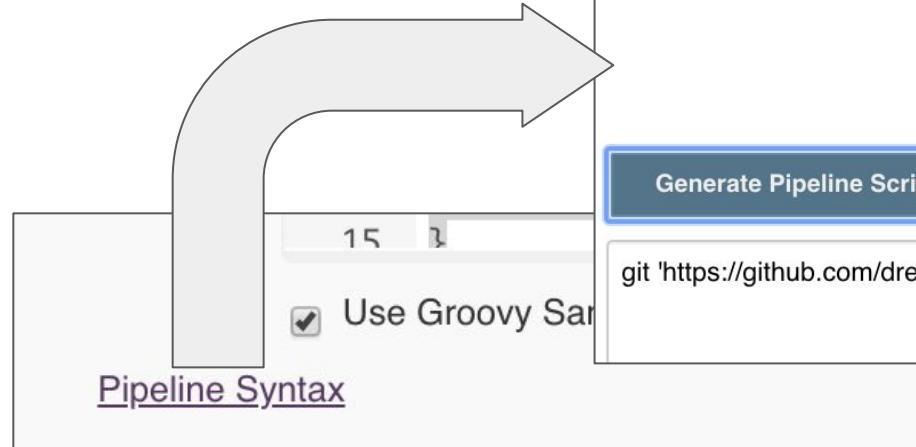
# Scripted Pipelines

- 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
  - Click on the Pipeline Syntax link
  - Fill in the resulting form
  - Click Generate Pipeline Script
  - It will give you the code



The diagram illustrates the process of generating a Jenkins Pipeline script. On the left, there is a large grey arrow pointing from a white rectangular box containing the text "Pipeline Syntax" to another white rectangular box. Inside the second box, there is a "Generate Pipeline Script" button, which is highlighted with a blue border. Below the button, the generated Groovy code is displayed:

```
git 'https://github.com/drehnstrom/mvn-messenger.git'
```

The second box also contains several configuration fields:

- Steps**: Sample Step git: Git
- Repository URL**: https://github.com/drehnstrom/mvn-messenger.git
- Branch**: master
- Credentials**: - none - Add
- Include in polling?
- Include in changelog?

# Declarative Pipelines

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

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL https://github.com/drehnstrom/devops-conve

Credentials - none - Add

Advanced... Add Repository

Branches to build

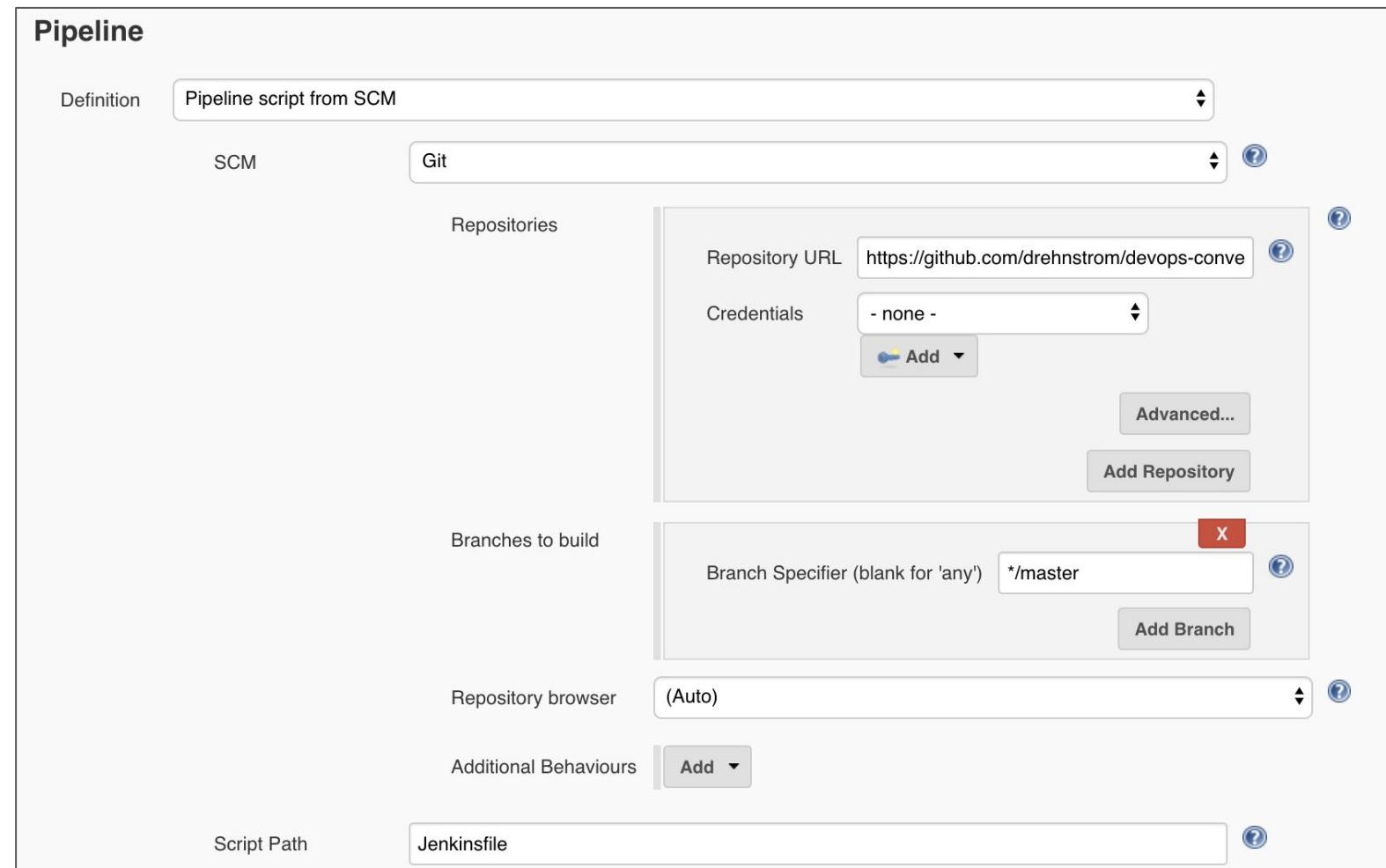
Branch Specifier (blank for 'any') \*/master

Add Branch

Repository browser (Auto)

Additional Behaviours Add

Script Path Jenkinsfile



# 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 two Jenkins pages side-by-side. On the left is the 'Build History' page, which lists recent builds. The first build (#26) has a dropdown menu open, showing options like 'Back to Project', 'Status', 'Changes', 'Console Output' (which is selected and highlighted in purple), 'View as plain text', 'Edit Build Information', 'Delete build #21', 'Git Build Data', and 'No Tags'. The 'Console Output' section on the right displays the log for build #26, starting with 'Started by user admin' and detailing the pipeline execution steps.

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
```

# Running Pipelines in Docker Containers

- Can use a custom agent in a Docker container to run your builds
  - Add the software required for your build into the Docker container rather than the Jenkins server itself

```
pipeline {  
    agent { docker { image 'python:3.7.2' } }  
    stages {  
        stage('build') {  
            steps {  
                echo 'Need to build the Program...'  
                sh 'python --version'  
                sh 'pip --version'  
            }  
        }  
        ...  
    }  
}
```

# 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
- SSH into Jenkins server and manually run steps from there
  - Helps discover missing components
- Output versions of required software from the pipeline
  - I.e., docker --version or python --version

# Tutorials: Jenkins

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

# Activity: Building a CI/CD Pipeline with Jenkins

- 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

# Chapter Summary

In this chapter, you have:

- Automated builds and deployments using CI/CD pipelines
- Used tools for analyzing code quality and testing
- 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 Code Build and Code Pipeline
- 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