Welcome to

GCP for Developers

DevelopIntelligence
A PLURALSIGHT COMPANY

Hello

About me...

- Training for the last 5 years
- Also train on React/Redux, Java, Kubernetes

We teach over 400 technology topics

We teach over 400 technology topics

You experience our impact on a daily basis!

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Objectives

At the end of this course you will be able to:

- Understand and work with Docker + Kubernetes/GKE + various services in GCP
  - We will give you real-world examples and exercises
  - Heavily exercise-oriented
- Also understand Terraform
  - This class is not a Terraform class - but you will walk away with the key points
- Understand a bit about the Autozone pipeline

Prerequisites

This course assumes you:

- Know some basic Linux commands. Nothing too difficult
- Have some background in software development

Agenda

- Intro to Microservices
- Intro to containerization and Docker
- Docker images/containers + Dockerfile
- Intro to GKE + Kubernetes - pods, services, replicasets, deployments, secrets (and best practices around that), configmaps, scaling, etc.
- Other relevant GCP Services - Cloud Functions, Pubsub,
- Walkthrough of Autozone pipeline
- Terraform overview/lab

Slide Deck

- The slides will be shared with you later today

Virtual machines and GCP accounts

- We have VM's with Docker/Terraform/K8s installed, your VM details here:
  - https://docs.google.com/spreadsheets/d/1u5_nN2CKHwsU1eZME6v8FIeFGQHUSpcU_IKd8xDRY6o/edit#gid=547734190
    - Be sure to open the https://training.datacouch.io/pluralsight/ URL in an incognito/private tab
- We also have GCP accounts for each of you, details here:
  - https://docs.google.com/spreadsheets/d/1Bz66yDVvRZHU6iuulX2g_LMJx1uMHA7KO78-Jgbjd2M/edit#gid=681413942
  - Use training@datacouch.io or support@datacouch.io as the recovery email address when it authenticates

Breaks

- We will take breaks every hour for about 7-10 minutes
- We will also take a 1 hr break for lunch around 12:00 Central time

Introductions

Student Introductions

Let’s get a sense for class knowledge and experience, I will go through the participants list

- Experience with Docker + Kubernetes?
- Experience with GCP?
- Role at Autozone?
- Fun fact

Installations

Installations

https://docs.docker.com/get-docker/

Docker Desktop comes with Docker, Kubernetes, and also kubectl (command line interface for Kubernetes)

Cloud Native:

microservices, infra as code, etc.

Definitions

Let’s start with 2 definitions:

1. cloud native
2. microservices

Cloud Native – what does that mean?

What does the word “native” mean to you in a technology context?

Type your answers in chat…

Cloud Native – what does that mean?

- Story time: when the cloud started to become popular in the mid 2000s (2005 or so) everyone started to throw their applications on the cloud using the lift and shift strategy - hoping to get the cost/performance benefits the cloud promised
  - Key takeaway - lift and shift == migrating to cloud w/ minimal changes to the application before the migration

- But they found they were often getting ***higher costs and worse performance***, so they soon realized that in order to get the benefits of running on the cloud they had to **first *re-architect*** *their applications to run effectively on the cloud*

- And hence the term ***cloud native*** was born - basically it means that if your app is cloud native then it is specifically architected to run on the cloud, and you can therefore get the benefits of **lower costs/better performance** that the cloud promises

Cloud Native – what does that mean?

- Long story short: cloud native == your application has been ***designed/architected to run on the cloud effectively***…

- “Native” used in other contexts like mobile app development
  - - ie, if we develop a mobile app maybe we only develop it for iOS and not Android, in which case we would say it’s ”native” to the iOS operating system..

- Basically ***native*** means its targets a specific platform

- Cloud native is an attribute of an application - if your application is cloud native it can then run effectively on the cloud!

Microservices + Monolith - definition

- To best define microservices, I think it helps to first compare it to the term ***monolith***

- A monolithic application is basically one gigantic application sharing a single codebase

- *What challenges would we face if for example, we were running all of Amazon.com in a single codebase - a single Github repository perhaps?*
  - ***Type your answers in chat!***

Monolith - disadvantages…

- **Deployment -** If we wanted to make any change - even a very small change - to the application we would have to redeploy the entire monolith
- **Shared libraries -** Also, if were to separate amazon.com into smaller pieces - say a cart and checkout piece, an item recommendation piece, a payment processing piece, etc - then it's safe to say that those pieces of the monolithic application would share libraries, so every time we make a change to one of those libraries, we would have to figure out what piece of the monolith is using that shared library so that we don't break the entire monolith by changing a small part of the library
- **Coding language restrictions -** Another constraint w. monoliths is that if our monolith starts as say, a Java application, then as we add additional pieces to the Java app they would need to be written in Java as well - which means you are limited by decisions made in the past.
- **Overall complexity -** It would also mean that as the complexity of amazon.com grows, it becomes harder to understand if it's managed as a single entity under a single codebase.

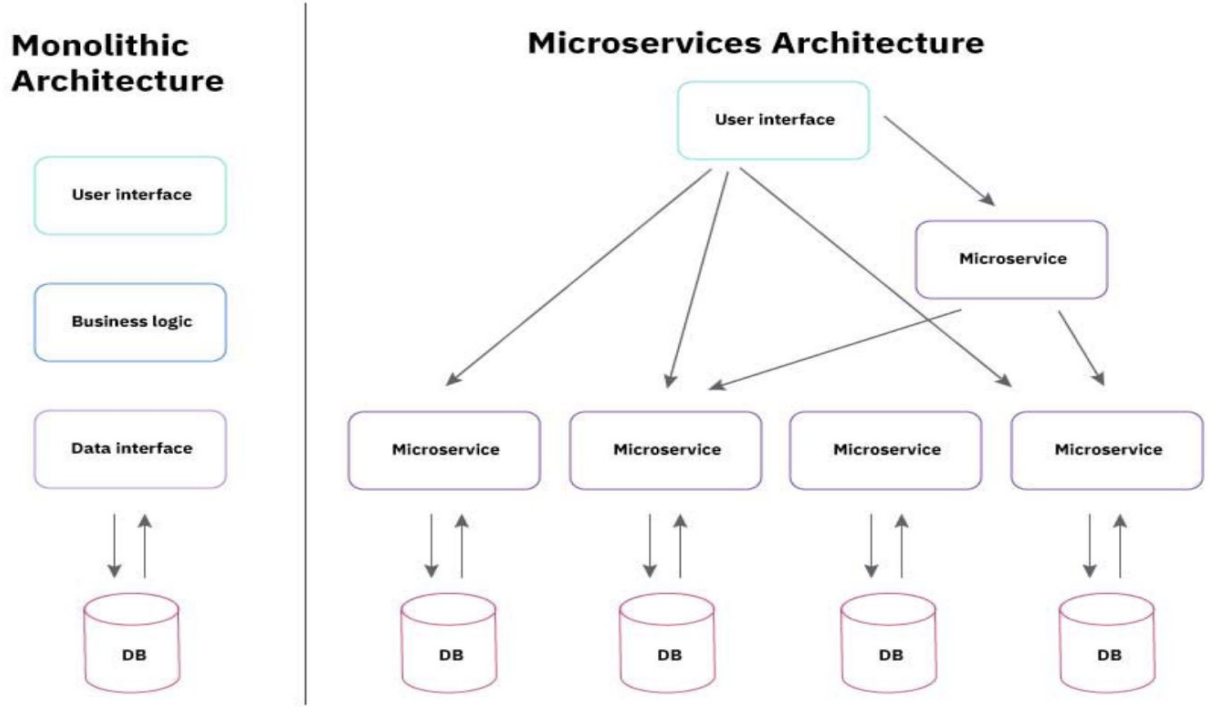
Monolith - solution…

- So, how do we fight the complexity of monoliths?
- Think: how do you eat an elephant?
  - One bite at a time!
- By breaking something down into more manageable pieces - **microservices** - we can better tackle the problem and deal with complexity.
- M**icroservices** are the “smaller pieces” of the larger application

Microservices example

- So, if amazon.com is our “elephant” we can have broken it down into smaller, more manageable pieces
- For instance, search could function as its’ own microservice, the cart would be its’ own microservice, etc.
- Each microservice would be its’ own application, with its’ own separate Github repository
- And all of these microservices combined would give us a fully functional amazon.com site!

Microservices vs monoliths…



Source: IBM blogs

Microservices characteristics

- Small, fully independent and autonomous application. Isolated from other MS's, maybe even with own database.
- Responsible for a logical part of a larger application "ecosystem". Singular responsibility.
- Can be any language. Developers on the same project can have completely different skill sets if necessary.
- Loosely coupled to other MS's. Communicate with other MS's using standardized protocols. Often a REST API or a message bus.
- People generally start with coarse grained services and then break them up into smaller and smaller pieces as the time goes by and they know more and more

Microservices benefits

- By dividing and conquering w/ microservices, if we want to make a small change to the site, we would just update the micro service that is responsible for that portion of the site
- And the key here is that we would only need to redeploy that particular microservice, without disturbing/interfering with all the other stuff that powers amazon.com.
- This is as opposed to using a monolith where we would have to redeploy all of amazon.com for a small change which doesn't make sense of course.
- *One of the clear advantages of using microservices is that it allows us to be **more agile***
    - *Agile means we can deploy quickly when changes need to be made, and respond to the needs of the business faster!*

Microservices - more benefits

- Small and manageable pieces. No massive monoliths.
- No need to have every microservice on the same language and framework.
    - For instance, w/ ecommerce app - your payment services can be in Python, and your user cart can in Java
- Testing is more limited since you don’t have to test an entire monolith.
- Can have their own development and release cycle.
- Can be deployed independently without breaking other pieces. Can be done faster.
- Independent scaling of MS’s. Faults are isolated. “Graceful degradation”.
- Encourages encapsulation of business logic.
- Smaller size enables Agile development.

Microservices - challenges

Now that you understand bit more about what a microservice is, what challenges can you think of w/ microservices?

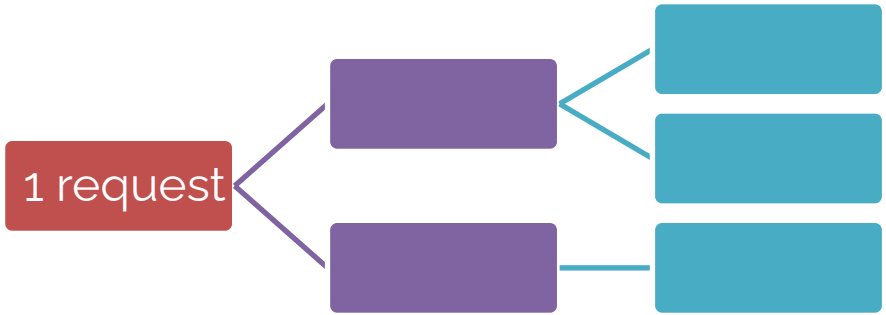
Type answers in chat…

Microservices - challenges

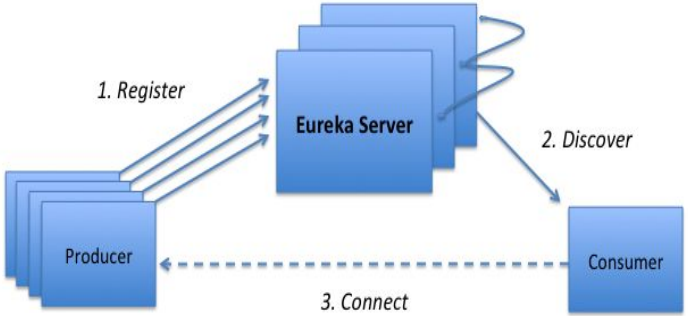
- Microservices come with their own challenges as well
  - Just like monoliths!
- A lot more moving parts. This often requires technologies that might not otherwise be used, such as message buses (like Kafka).
- Requires seasoned architects and leads to keep disaster at bay.
- Often requires more infrastructure to support, as each component has its own overhead.
  - With a monolith, overhead can often be shared.
- Coordination between dependent MS’s can be more complex than a monolith.
- More parts to deploy.

Microservices - challenges

- Service Discovery (need Service Registry)
- Fault tolerance
- Integration Testing
- Requests can fan out



Service Registry:



Microservices: Discussion

E-commerce application

DevelopIntelligence

A PLURALSIGHT COMPANY

Microservices discussion - breakout rooms

- Think about Amazon.com - where each web widget on the homepage can be backed by hundreds of microservices. Think about microservices for personalization of home page & personalization of search.
- Pick a specific web widget from the homepage - it could be search bar, cart, item recommendations, sign in, etc…
- Now, how you would break this down into different functionalities? What are the microservices supporting those functionalities? How do they communicate? How is data stored for each service?
- Take 10 mins to think about it with your team and have one person from your breakout room share your findings.

Microservices discussion

Major functionality:

- Product catalog (admin functions, search, recommendations, related products, etc.)
- Shopping cart (and email reminders about items in cart)
- Order processing (payment processing, emails, etc)
- Customer order cancel and refunds
- Fulfillment (send to warehouse, inventory allocation, picking, shipping, backorders, etc.)
- Reporting (internally and to the customer)
- Email marketing
- Social media integration
- Shipment tracking
- Bonus: Digital content! Movies, music, etc.

Stateless applications - overview

- No session state saved on a server.
- State is stored on external system in db, cache etc.
- Enables disposable infrastructure – servers can be trashed without concern for botched user sessions.
  - Becomes important w/ Kubernetes
  - Enables rapid scaling – both out and in
- Enables fast recovery from server failures. Users aren't stuck to a failed server.
- Enables easy and fast deployments.
- Session state stored on client and passed with each request. Example: cookies, JSON Web Tokens

Infrastructure as code - overview

- Code for the management of infrastructure (networks, servers, load balancers, volumes, orchestrators, etc)
- Provides automation and control.
- **Allows versioning since code can go into a code repo.**
- Solves configuration drift.
- Can prevent/overwrite manual interventions. Enforces consistency.
- Allows code review of infrastructure. Imagine the benefits to security conscious organizations (which should be all organizations).
- Allows fast build-out of identical environments. Also allows you to test environment changes before changing production.
- Several technologies exist such as TerraForm, Cloudformation, Ansible, Chef, Puppet, and many more.

Infrastructure as code - overview

- In this class, we will also see how Docker helps us w/ Infrastructure as code

Auto-healing overview

- What does the term auto healing mean in terms of software system?
- Autohealing **lets you automatically restart apps that are compromised**.
- It promptly detects failed instances and recreates them automatically, so clients can be served again.
- With autohealing, you no longer need to manually bring an app back to service after a failure
    - With Kubernetes autohealing is built in

What is auto-scaling?

- If our application gets a sudden increase in traffic, what do we need more of?
- ***Type answers in chat***

Auto-scaling overview

- Scale horizontally vs vertically. What does this mean?
  - Analogy: retail store during Black Friday
    - What do stores have to do during holiday season?
- Scale in/out - horizontal, out = increase horizontally, in = decrease horizontal
- Scale up/down - vertical, up = increase vertical, down = decrease vertical
- What are some metrics that would trigger scaling?

Cloud Native

Making it a reality

How do I make my application cloud-native?

- ***cloud native*** == your application has been ***designed/architected to run on the cloud effectively***…

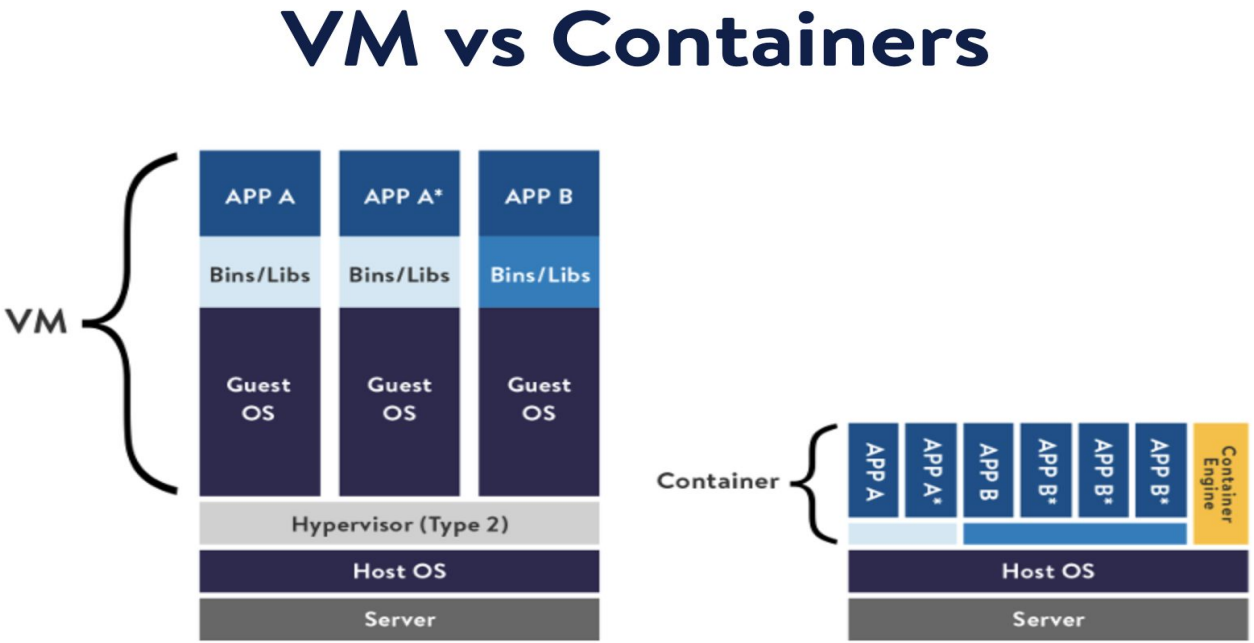
- We've covered **what** cloud native means but now the question is ***how*** can we make our application cloud native?
  - What specific steps can we take to make an application cloud native?

Cloud native - characteristics

- **The Twelve-Factor App:** Baseline best practices for designing & building web applications for portability and resilience.
  Reference: https://12factor.net/
- **Microservices:** Independently deployable services that do one thing well (Single Responsibility Principle).
- **Self-Service Infrastructure:** Platforms for rapid, repeatable, and consistent provisioning of app environments and backing services.
- **API-based Collaboration:** Published and versioned APIs that allow interaction between services.
- **Anti-Fragile:** Systems that are resilient to stress and failure.

Docker Containers

Containers vs VM's - what differences do you see (answer in chat)?



Containers vs Virtual Machines

- Both VM's and containers are essentially "wrappers" around our applications making it easier to run our apps on the cloud
- Containers allow us to share a copy of the OS between applications
- Apps running inside a VM each have their own copy of the OS
- Hence, VM's are heavier
- Containers are lightweight, low overhead, and we can run more apps on a single server w/ containers
- Sidenote: VM's offer better security than containers due to OS isolation

Docker-specific Terminology

- Docker ***images*** are how we share our application
- An instance of a Docker image is a running container….this is a running instance of your application!
- DockerFile + your application code -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- Docker ***images*** are how we share our application
- An instance of a Docker image is a running container….this is a running instance of your application!

What problem does Docker solve?

- Concept of containers not invented by Docker
- But Docker executed the concept very well!
  - Helps eliminate the ‘works on my machine phenomenon’
  - Docker containers are consistent across different environments
  - Simple example: trying to get a Node.js app to work locally is tough – containerizing it makes it run consistently on different machine setups!
- The Docker ecosystem is excellent….

The Docker ecosystem

- Docker is more than just containers – it's really a set of products to make it easy to share, run, and create Docker containers
  - Hence, "ecosystem" is appropriate
- Docker images – shareable templates for your applications
  - Analogy: Java class - more to come on images…
- Docker Desktop – easily run Docker images locally
  - Now with Kubernetes!

The Docker ecosystem part 2

- Docker CLI – easily run images locally, create new images, and manage images
- Docker Engine – enables containers to run consistently on any infrastructure – various Linux distros + Windows/Server
  - https://www.docker.com/products/container-runtime
- Docker Hub – easily share images with the world

Docker Images

Docker images

- An image is a template for containers

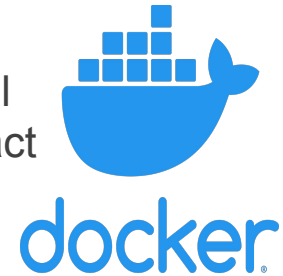
- Good analogy: Docker image is like a Java class – where a Java class is a template for creating Java objects, Docker images are templates for actual containers
- run an image -> gives you a container
    - one image -> can create multiple containers
- implement a blueprint -> gives you a house

Docker images - details

- Images are immutable – can ***not*** be changed
- If you want to make some changes to your application or Dockerfile then you would have to create a new image

Docker images - what's inside?

- Think about any application you've worked on in the past
- What would you need to run that app on a server that's not your local machine?
  - The source code, the dependencies, tools, etc..
  - All of that stuff is inside a Docker image!
- For instance, a Node.js + React application – what's inside that?
  - Our ReactJS files – our frontend
  - Our Node.js runtime – our backend
  - Our Node.js web framework – Fastify.js, Express
  - Other JS dependencies/libraries – like Babel, Redux, etc..
  - Also the Alpine OS – CentOS, Ubuntu – all share the Linux kernel
  - All of this is part of the Docker image created for our Node.js/React app!



Docker images - layers

- A Docker image is comprised of many layers internally
- You may also notice this when pulling an image – Docker will create a separate line in your command prompt for each layer it pulls for that particular image
- The layers within an image may look something like this:

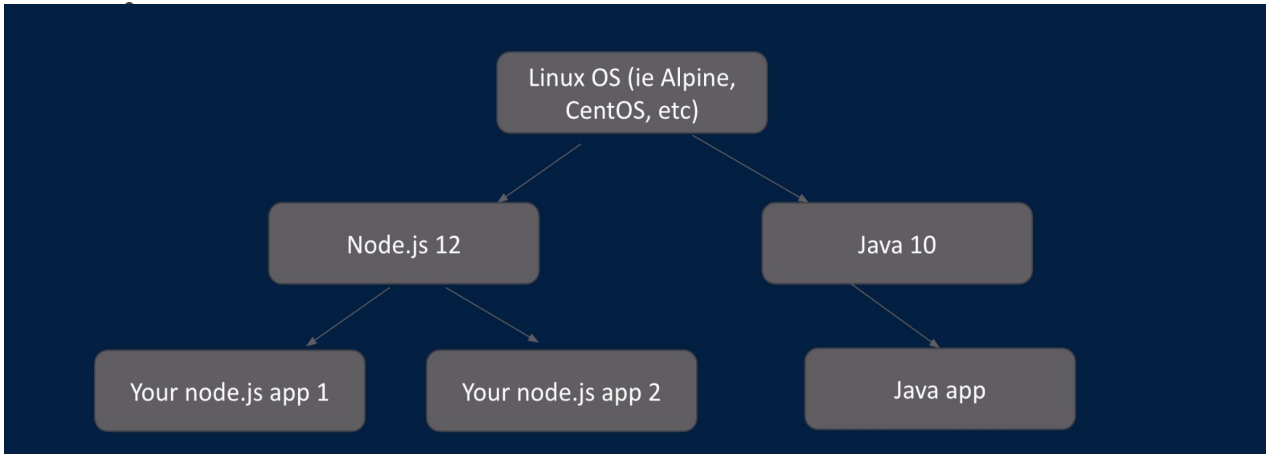
Layer 1: Linux OS: Alpine ~ 50 MB
Layer 2: Security fixes: ~ 2 MB
Layer 3: Node.js runtime: ~ 30 MB
Layer 4: Your Javascript files: ~10 MB

- When pushing/pulling images, only changed layers are are pushed/pulled to save on Bandwidth

Docker images inheritance

- Images typically inherit from other images
- This is like a class hierarchy in an OOP (like Java)
  - You can also override functionality in an image that you inherit from

- Where do we specify which image we want to inherit from?
  - In the Dockerfile! We'll see more later…



Docker CLI

Docker CLI - overview

- The Docker CLI looks a lot like Linux commands
- All commands start with “docker”
- **When working with the Docker CLI, there’s really 2 sets of commands:**
  - one set for working with ***images***
  - One set for working with ***containers***
- Let’s cover the CLI for images first…

Docker CLI - containers

Docker command line interface gives us everything we need to build, interact, inspect, and run containers and images. Many of the commands mimic Linux commands. All commands start with “docker”.
Reference: https://docs.docker.com/edge/engine/reference/commandline/docker/

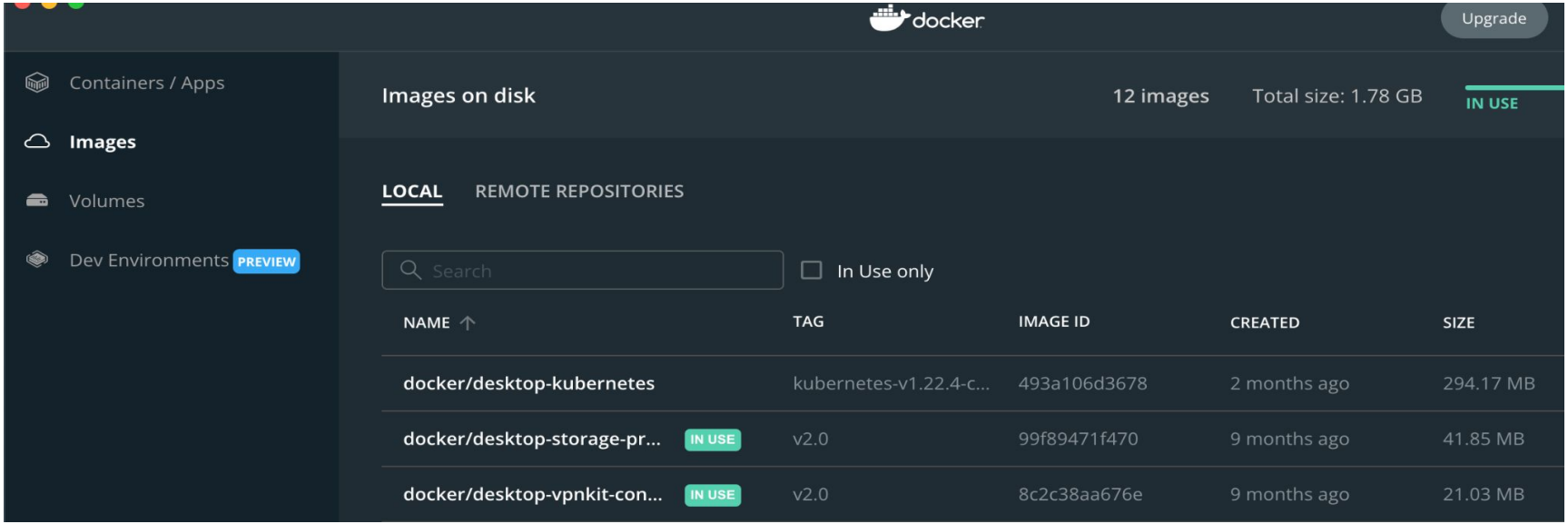
- Containers:
  - ps – list running containers (include –a to see stopped containers)
  - rm – remove container
  - inspect – inspect running container
  - start – start a stopped container
  - stop – stop running container
  - restart – restart a container
  - cp – copy files to/from a container to local filesystem
  - logs – show the console output from a container (include -f tailing the log)

Docker CLI - images

- Images:
  - build – build an image from Dockerfile
  - images – show images
  - rmi – remove image
  - run – run a container from an image
    - -d – detached
    - -p – port mapping
    - -it – interactive (gets you a command line in the container)
    - --name – assign a name
    - -e – environment variable
    - -v – volume mapping
  - push – push the image to a repo
  - pull – pull an image from a repo
  - tag – tag an image

Docker Desktop

- Much (but not all) of what we do through the command line can also be seen/done in Docker Desktop
- Docker Desktop:



Docker ps (containers)

- To view all running containers you can use “docker ps”
- To view all running **plus** stopped containers you can do “docker ps -a”
- What is a stopped container?

Docker run command (images)

- ***Docker run*** is a command you will use a lot
- ***Docker run*** can be used to run a Docker image, which gives us a container…
- Remember an image is just a template for an application, and a container is a running instance of that template
    - Like Java class->Java object
- For instance, we can do **docker run nginx**
- This will give us a Docker container

Docker CLI - port mapping

- By default, when you run a container with docker run, it doesn't publish any of it's ports to the outside world – ie, your localhost when running Docker Desktop
    - Which makes sense because a container is it's own isolated environment!
- So, how do we expose that internal port?
    - Through the '-p' flag (p for publish) when we do a **docker run**
    - So, if we do **docker run –p 8080:80** that says map port 80 in the container to port 8080 on the Docker host – your localhost in this case
- docker run -p [host_port]:[container_port] [docker_image]

Docker run command - overriding

- We can override the default command that is run by the image by specifying an extra parameter
- Example: ***docker run busybox***
  - This just starts the container
- But if we do "***docker run busybox ls***"
  - It will start the container and list the contents of the current directory in the container

Docker run - naming containers

- When creating a container, you can specify a name for the container
- If you don’t specify a name, Docker will give the container a name automatically
- Docker has fun with the name when it generates it (if you don’t specify one)
  - See Docker Dashboard!
- To specify a name for the container, simply do this:
  - `docker run --name mynginxcontainer`

Docker run - pulls images

If you try to use docker run with an image that you have not already pulled locally, it will pull it!

If an image doesn't exist on the host yet, it will be pulled before running.

Run is a shortcut operation for:

- Docker pull <image>
- Docker create <image>
- Docker start <container>

example: docker run -it -p 27017:27017 mongo:4.0 /bin/bash

Docker run - foreground/detached modes

- **When running a container it can run in either foreground or detached mode**
- Foreground mode is the ***default***
  - This is why you get the container output in the terminal:
- Foreground - when running a container console executing docker run attaches to standard input, output and error of the container:

```
varoonsahgal@Varoons-MacBook-Pro ~ % docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
```

In detached mode the container runs in background.
docker run [docker_image]
docker run -d [docker_image]
docker run -it [docker_image] [shell]
example: docker run -it -p 27017:27017 mongo /bin/bash

Docker run - detached mode

- In detached mode the container runs in background.
- Use the - d flag to specify detached mode:

```
docker run -d [docker_image]
```

- foreground runs by default:

```
docker run [docker_image]
docker run -d [docker_image]
```

```
docker run -it [docker_image] [shell]
example: docker run -it -p 27017:27017 mongo /bin/bash
```

Docker exec command

- ***Docker exec*** can be used to **execute** any arbitrary process available in a ***running*** container.
- Note that you can NOT use docker exec with an **image,** just a **container** (running instance of an image)
- Example of it's usage: you can use **docker exec** to "log in" to a shell. This is dependent on the image the container was built from.
- For example: the alpine shell is /bin/ash
- This is much like an SSH connection to a server

Docker exec command

- Once inside the container, you can look around and inspect your container
- This is very useful for debugging.
- Example of why: Suppose you've created an image and you're not sure if the image looks like you expect. So you can get into a running container and view things.
- For example, when we built the nginx image, how did we know the file was there until we tried the web site?
- The location of the shell process depends on the container.
- ***docker exec -it nginx bash***
    - This will open up a shell in the nginx container

docker run vs docker exec

- Docker exec is NOT meant to start a new container from an image
- For instance, if you try to run **docker exec nginx**
  - You will get an error
  - However, **docker run nginx** works just fine
- Docker exec is meant to execute a process in a running container
  - So it’s good for creating a shell inside a container!
- Docker run is meant to run a new container based on an image

Mutable vs immutable infrastructure

Mutable:

- You would ***upgrade*** the software being used on a server to a newer version if necessary
- For instance, w/ Chef can run a Chef “recipe” if you want to upgrade version of nginx used by server to newer version

Immutable

- You would ***delete*** the software being used on a server and then create a newer instance with the correct version if necessary

- For instance, w/ Terraform to create a new instance of nginx you would **throw away** the old instance w/ older version and then create a newer one w/ the correct version

Docker Exercise

Docker exercise 1

- To best understand Docker, it helps to play with it!
- Let's create our own Docker image by pulling something from Docker Hub
- Let's try Wordpress: https://hub.docker.com/_/wordpress
- Read the instructions
- Your goals:
    1. "Pull" and then Run the image locally. Use the docker pull and docker run commands and a port mapping. Remember images must be run to get containers - images -> containers
    2. View the Wordpress UI once the container is up and running
    3. Try creating another Wordpress container while you still have this one running – remember not to use the same port on your localhost!
        1. Also, try to view the new instance of Wordpress in your browser
        2. This should help consolidate that a single image spins up multiple containers – becomes more important w/ K8s!

Docker exercise 2

Quick lab: go ahead and use docker run for an image that you do NOT have locally:

docker run tomcat == docker pull and start

Note that this will pull the tomcat image, and also start a container based on that image automatically

So, you did not even have to run docker pull!

Docker Terminology revisited

- Docker ***images*** are how we share our application
- An instance of a Docker image is a running container….this is a running instance of your application!
- DockerFile + your application code -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- Docker ***images*** are how we share our application
- An instance of a Docker image is a running container….this is a running instance of your application!

Dockerfiles and tags

Docker tags

- In Docker, we can tag our images to give them some metadata
  - Could be a version number, what OS they are meant to run on, etc
- Let's look at an example here for Jenkins tags: https://hub.docker.com/r/jenkins/jenkins/tags?page=1&ordering=last_updated
- To tag an image, you can run **docker tag httpd:1.2** as an example of adding a tag of 1.2
  - After the colon, we specify the tag
- Every image will have one tag by default if none is specified

Docker tags pt 2

- What if we don’t specify a tag for an image?
- By default, if you omit a tag, Docker will give it a tag name of **latest**
  - But **latest** is misleading, because it doesn’t always mean that’s the latest version of the image
  - Example - developer created an image, didn’t tag it, so it’s given the “default” tag, which is “l**atest**”
  - then developer created a newer image and tagged it 1.2 – the newer one has a tag of 1.2, but ***latest*** will actually refer to an older image!

Dockerfile

- Remember that **Dockerfile + your app source code -> Jenkins (runs docker build) == Docker Image**
- Dockerfile is how to create Docker images

- A Dockerfile is an example of Infrastructure as code
  - *Infrastructure as code (IaC) means to manage your IT infrastructure using configuration files.*
  - Before IaC, IT admins would have to ***manually*** *change configurations in order to manage their infrastructure*

Dockerfile - more

- The commands in a Dockerfile execute from top to bottom
- And each command is also independent of other commands
- Commands are relative to the directory in which the Dockerfile lives

Dockerfile - more

- The commands in a Dockerfile execute from top to bottom
- And each command is also independent of other commands
- Commands are relative to the directory in which the Dockerfile lives

Dockerfile build command

How do we create a new Docker image?

- By building a Dockerfile!
- **docker build –t <imagename:tag> <directory>**
  - ”t” is to specify a tag

Example:

- **docker build -t myimage:v1 .**
- Where **<directory>** is wherever your Dockerfile lives
- Inside the **Dockerfile**, there is a reference to your app code
- We will be using this command shortly…

Dockerfile commands

A sampling of commands you may see in a Dockerfile:

- FROM – which image you want to inherit from
- RUN – Run a Linux command in a new layer. The command is dependent on which shell is in the image.
- EXPOSE – Tells what ports this container will listen on. Does not actually publish a port, but is rather like documentation.
- COPY – copy files into the container.
- ADD – copy files into the container (same as COPY, but supports 2 other sources - additional URL source, tar extraction from local)
- ENV – Define environment variable
- ENTRYPOINT – The command to be run when container is started. This is inherited from a base image.
- CMD – provides defaults to be run after the ENTRYPOINT. Also inherited from the base image

Docker Exercise - Dockerfile

Dockerfile lab

- Visit https://hub.docker.com/_/httpd and see instructions on how to use this image
- Let's grab the Docker image for Apache – a popular web server:
- Docker pull httpd //technically you don't need to do this
  - By default this will pull the latest image
- Then run Apache locally:
  - docker run -p 8080:80 httpd
- Then visit http://localhost:8080 to see a page that says "It works"

Docker lab 2 - Apache success

Docker lab 2 part 2 - Dockerfile

- Now, we have an Apache server running locally
- But how would we serve up our very own html files in the apache server?
- We can use a Dockerfile for that!
- Let's create our own Dockerfile using the httpd/Apache image we just pulled as a base image
- Remember: A **Dockerfile** is a text document that contains all the commands a user **could** call on the command line to assemble an image.
- So, the point of a Dockerfile is to help us create an image for our specific application

Docker lab 2 part 2 steps

Let's take these steps:

1. Create a file named **Dockerfile**
    1. You can do this anywhere really, don't technically need a repo – best to create a folder
2. Inherit from the base Apache image we just pulled – simply use "**FROM httpd:latest**"
3. Add a file named test.html so that we can give Apache something to actually serve up instead of the "it works!" text. The **test.html** would be our "application code" in a real application.
    1. You can put this in the **Dockerfile**:

    COPY ./test.html /usr/local/apache2/htdocs/

        1. So, no need for the the public_html folder as mentioned in the docs here: https://hub.docker.com/_/httpd
4. Now, build it! Use the **docker build** command:
    1. **docker build –t <imagename:tag> <directory>**
    2. **What happens when you leave out the tag?**

5. Now that you've built the image – run it with **docker run -p [host_port]:[container_port] [docker_image]**

Try exposing it on a different port – the instructions say 8080, but try 3500 or something else. If you visit localhost:3500/test.html you should see your page!

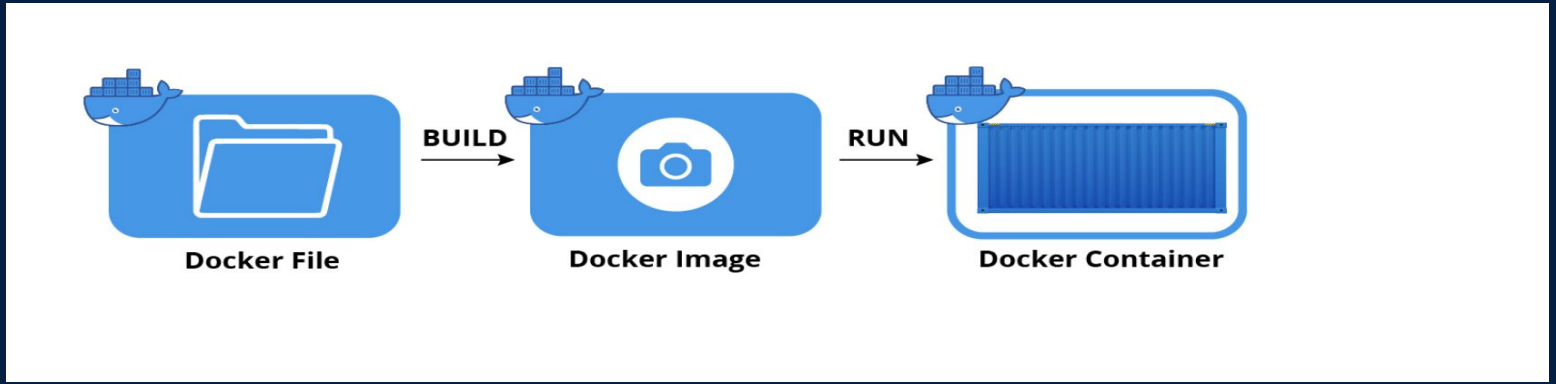
Docker lab 2 part 3

6 . Let's now see how to bring up a bash shell so we can see what's going on inside the container: **docker run –it myfirstimage /bin/bash** or **docker run –it myfirstimage /bin/sh** (if bash not part of image)

Often, you will want to step inside your container to see what's going on. This is how you would do that!

7. Go inside the **/usr/local/apache2/htdocs** directory and you should see the test.html file that you just copied over!

Docker terminology revisited



- DockerFile + your application code -> Jenkins -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- Docker ***images*** are how we share our application
- An instance of a Docker image is a running container….this is a running instance of your application!

Docker Networks

Lab: Docker container isolation

Docker containers are isolated by default.

Validate this by doing a very simple exercise:

Open a terminal window and then do:

docker run -it busybox sh
touch helloworld

Then, open a new terminal window and create another container w/ the same commands:

docker run -it busybox sh
ls

What do you see as the output of “ls”? Do you see the helloworld file we created?

Think:

Suppose I have an application - like Wordpress.

The application needs a database to store information about the blog - specifically it needs and uses the MySQL database to store blog posts.

Wordpress will often need to write to the MySQL database - so it needs to communicate with MySQL

So, if I have a Wordpress container and a separate MySQL container - how can those containers communicate in Docker??

Answer: ***Networks!***

Linking - Legacy feature

- Networks are the way to go, but you should know a bit about linking….
- This is a ***legacy*** feature, but still useful to know.
- Networks (next) are the preferred way of communicating.
- You can explicitly link containers like wordpress and mysql to each other so that they can communicate by name.
- You link containers by including a --link option in the run command and provide the container name.
- Linking does not scale well - try linking 10 services with each other!

Networks in Docker

- You can create Docker networks to allow containers to easily communicate.
- Types:
  - Bridge (default) – this one is created by Docker and containers go here by default. Not recommended for production. No container communication by name.
  - Bridge (user-defined) – Recommended for standalone containers running on the same host. Containers can communicate by name.
  - Overlay – allows networking across Docker hosts. Works with Swarm.
- Networks are managed with the docker network ... commands
- docker network will show all commands, example **docker network ls** will display all existing networks including 3 pre-defined networks.
- Containers are put into a network with the --network option in the docker run command.
- Existing containers can be put into a network with **docker network connect <network name> <container name>**

Networks in Docker

- You can create Docker networks to allow containers to easily communicate.
- Types:
  - Bridge (default) – this one is created by Docker and containers go here by default. Not recommended for production. No container communication by name.
  - Bridge (user-defined) – Recommended for standalone containers running on the same host. Containers can communicate by name.
  - Overlay – allows networking across Docker hosts. Works with Swarm.
- Networks are managed with the docker network ... commands
- docker network will show all commands, example **docker network ls** will display all existing networks including 3 pre-defined networks.
- Containers are put into a network with the --network option in the docker run command.
- Existing containers can be put into a network with **docker network connect <network name> <container name>**

LAB - Docker Network

Containers are isolated by default, so you can put them in a docker network to share access

Let’s connect Wordpress to MySQL by using a ***user-defined*** Docker network!

LAB - Docker Network

1. You'll need a MySQL Docker container to be the DB for WP site
2. Containers are isolated by default, so you can put them in a docker network to share access - **docker network create my-wp**
3. Create the MySQL container and specify some setup like DB name and login info: **docker run --name wp-mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=wp -e MYSQL_USER=user -e MYSQL_PASSWORD=admin --net=my-wp -d mysql:latest**
4. Run a WP container that connects to the mysql container: **docker run --name wordpress -e WORDPRESS_DB_HOST=wp-mysql:3306 -e WORDPRESS_DB_USER=user -e WORDPRESS_DB_PASSWORD=admin -e WORDPRESS_DB_NAME=wp --net=my-wp -p 8011:80 -d wordpress:latest**
5. Finish setting up the site and confirm you can make posts!
6. Remember: the DB data is (currently) only stored in the container, so if you delete the container your data goes away!

Advanced Containers - Docker Compose

Docker Compose

In the last lab, we saw how to create a docker network manually. Imagine if we had more than 2 containers - the command could get quite long!

Is there a simpler,cleaner way to create a network? Answer: ***yes, with Docker Compose***

- Compose is a lightweight way to run your application without the overhead of a Swarm or Kubernetes application.
- Compose is a tool for running multiple related containers together with one command. It is included with Docker by default.
- So, you don't have to start each container separately

- You specify all the configurations in **docker-compose.yml** file.
    - This is another example of ***infrastructure-as-code***!

Docker Compose

- compose creates a new user defined bridge (just like we did manually in last lab) network for your containers
- This way they can communicate with each other by name. Or you can specify your own.
- You can refer to an existing image in Docker Hub, on your local machine, a private repo, or you can refer to a Dockerfile location, and ***Compose will build the image for you before running the container.***
- To start up your app, use **docker-compose up**.
- To stop it, use **docker-compose down**.
- It will automatically look at the **docker-compose.yml** file

Docker Compose - sample docker-compose.yml file

```
version: '3.3'
services:
  wordpressapp:
    build: path/to/Dockerfile
    image: “db:2.0"
    restart: always
    ports:
    - "8089:80"
  mysqldb:
    image: “db:2.0"
    restart: always
    ports:
    - “3306:3306"
    environment:
    - MY_ENV=my_value
```

Compose will do 3 major things:

1. create a network so our containers can communicate
2. create the image (by building the wordpressapp Dockerfile), and then run that image to create the container
3. create the container from the mysqldb image

docker-compose down

- this command **will stop running containers**, but it also ***removes the stopped containers as well as any networks*** that were created.

LAB - Docker Compose

- Create a new Docker Compose file
- Use two Docker Hub images. One for Wordpress, and the other a MySQL image.
  - Review the parameters we passed in when we ran the command manually in the last lab
  - You'll need to review the documentation for the images to find out what important options to pass the containers.

Bring the stack up using **docker-compose up** and try to use Wordpress blog

Advanced Containers - exec, it

Interactive Shell

- Containers are mostly intended to run as headless without interaction.
  - headless == no interface, not bash shell, etc..
  - Sometimes however you may want to interact and provide input
- To interact, you can run attached or attach later to interact with it.

  docker run -it <image-name>

  docker attach <container-name>
- You can now get it's output (like with docker logs), provide input if necessary.
- If you attach at run then to detach without stopping it, use the detach sequence (hold CTRL, press p then q), CTRL+ C will end the container process.
- We will see how to 'login' in the container next, and how to install software

Exec command

- With a **running container**, you can execute any arbitrary process available on the container using **docker exec**
- Frequently you can use this to “log in” to a shell. This is dependent on the image the container was built from. example: /bin/sh or just bash
- This is much like SSH connection to a server
- Once inside, you can look around and inspect your container
- This is very useful for debugging. You’ve built an image and you’re not sure if the image looks like you expect. So you can get into a running container and view things.
    - We saw how to do this earlier with docker run -it <image name> /bin/sh
    - differences between run and exec in next slide
- The location of the shell process depends on the container.
- **docker exec -it <container name> bash**

Docker run vs docker exec

- docker run:
  - runs an image to create a container
  - you can specify you want to create a shell to connect to the running container by doing docker run <imagename> /bin/sh
- docker exec:
  - can NOT use docker exec with an ***image***
  - has to be a RUNNING container
    - can’t be a stopped/paused container
  - Can connect a shell to a ***running*** container:
    - so if you do “docker ps” you can find the container name/ID, then do a “docker exec <container name> /bin/sh
- To summarize: docker exec is to be used with running containers, docker run is to be used with images (and it also pulls images if they don’t exist locally)

LAB - docker exec

- Start a new container from the nginx image on Docker Hub
  - Use docker run but don't specify the -it param
- Using exec, get into the container and have a look around

  docker exec –it <container name> bash
- Visit the web site and make sure you see the default home page.
- Now go change the home page index.html file located in /usr/share/nginx/html/
  - Wait, how do you edit the file?
  - Try installing vim. This is a Debian OS, so use **apt-get install vim**. First update the package manager with **apt-get update**
    - apt – advanced package tool – which is used to install/remove software on various Linux distros
  - Now update the headline
- Verify that your changes show in the web site.
- Let's discuss "How will it impact the image?"
- Let's discuss "How do I persist the changes?"

Registries

DevelopIntelligence
A PLURALSIGHT COMPANY

Docker registries

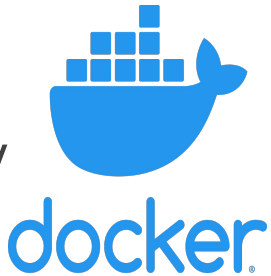
- We want to share our applications with the world – how do we do that?
  - With images – which encapsulate our application
- But where do we put those images?
- Inside a Docker registry!
- **Docker registry** is open source software that you can use to set up your own registry
  - You can see how to set one up here: https://docs.docker.com/registry/
  - ***Anyone can set up their own Docker registry***
- Cloud providers also provide their own registries for storing images, like Azure, GCP, etc..

Docker hub

- **Docker Hub** is a specific type of registry from Docker w/ additional features like teams, orgs, etc…
- Within Docker Hub you have many repositories, which each represent a distinct application!
  - There's repositories for Jenkins, Apache, nginx, etc – basically any application that wants to share it's "Dockerized" binary – it's image.

Docker repositories

- Within a registry, you would have many repositories
- In Docker, a repository is a collection of images of the same application
- Basically, a repository is a place where multiple versions of a single application live
  - For example, there is a repository for a Jenkins image:
  - https://hub.docker.com/r/jenkins/jenkins
  - Repos can be public or private on Docker hub, just create an account..
- To summarize, smallest to largest: image -> repository -> Registry -> Docker Hub (Registry + Extras)



Docker terminology summarized

- To summarize, smallest to largest: image -> repository -> Registry -> Docker Hub (Registry + Extras)

LAB - private registry

Let's simulate having a private registry by starting one on our local workstation and then work with it. It runs as a Docker container!

1. Find the registry image on Docker Hub. Review the instructions and run it on your machine.
2. Now try tagging an image and pushing it to your new private registry. Perhaps you can pull Apache/nginx/busybox from Docker Hub and retag it? Try it.
3. Access the REST API to look at the catalog of images `http://localhost:5000/v2/_catalog`

Note that there's more involved with setting up a production grade registry, such as authentication. If you're working with a secured registry, you just need to login with docker login.

Advanced Images

Images - creating from containers

- There's two ways to create a Docker image:
  - Dockerfile
  - create images from running container
    - This is like going backwards! Normally: Dockerfile -> image -> container, but for now: running container -> image

- We can actually create Docker images from containers (***without*** using Dockerfiles), but can you think of a reason why we would **not** want to do this?

- Type answers in chat…

Images - creating from containers

- The primary reason not to do it this way is b/c we have then moved away from infrastructure as code (IaC)
  - b/c the Dockerfile is our infra as code - it can be version controlled and it tells us what base images we are using, what their respective versions are, what modifications have been made, etc..
  - when we create an image from a container, we can't look at the Dockerfile to tell us exactly what changes were made for that particular image
- However, doing this can be useful sometimes…

Images - creating from containers

- This can be useful for small experiments and proof of concepts

Steps to create an image from a container…

The general procedure roughly looks like (docker run does both of these):

docker create … (to create a new container)

docker start … (this starts the container you just created)

NOTE THAT DOCKER RUN DOES BOTH CREATE AND START

Make changes in the container, install software, etc.

docker commit … (this creates a new image without a tag)

- This command is new to us
- How we create an image from a container

LAB - create an image from a container…

Let's create an image from a container. We'll start with a base image from Docker Hub. Then customize it, and finally create an image from that.

1. First, create a new container using the Nginx image
    - docker run -it nginx bash
        - this will create a new container and also create a bash shell for you
2. Make some changes to the home page
    - The homepage is at /usr/share/nginx/html/index.html
    - How do you edit the file?
    - Try installing vim. This is a Debian OS, so use `apt-get install` vim. First update the package manager with `apt-get update`
    - Now update the headline

3. Then, create a new image from that container. You can name it "mynginx" AND give it a tag of v1

    Use the **docker commit** command - read the docs: https://docs.docker.com/engine/reference/commandline/commit/

4. Now, delete the old container.

5. Finally, create a new running container from your new image (mynginx:v1) and verify that it serves the home page you updated - which lives at /usr/share/nginx/html/index.html

Advanced Images multi-stage builds

Multi-stage builds

- We want to keep the size of our images down
  - Because if we have maybe 1,000 instances of our container, keeping size small as possible matters even more!
- Multi-stage builds help us do that..
- How does it help?
  - Because it makes it easier to get rid of any unnecessary build artifacts that may be created during the build process
    - sometimes intermediate artifacts like Java/GO/etc SDK and other things are leftover in the build
    - We do NOT want that stuff leftover in our final image, its not necessary and takes up space

Multi-stage builds - stages

- A Dockerfile is multi-stage if you see multiple FROM statements:
- This example below has 2 distinct stages
- The second stage references the first stage by using **COPY –from=0**
  - **0 is the first stage**
  - this says just grab what i need from the 0th (first) stage, then put it inside an Alpine OS

```
# syntax=docker/dockerfile:1
FROM golang:1.16
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go ./
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app ./
CMD ["./app"]
```

Multi-stage builds - naming stages

- We can instead name stages using “AS”
- Easier to read, and makes more sense especially if the instructions in your Dockerfile are re-ordered later, the COPY doesn’t break:
- reference:https://docs.docker.com/develop/develop-images/multistage-build/

```
# syntax=docker/dockerfile:1
FROM golang:1.16 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go    ./
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app ./
CMD ["./app"]
```

Images - ALPINE

- A goal of image creation should be reduction of image size.
- Some images can be very large. Many images in Docker Hub exceed 1gb.
- Additionally, many standard distros contain security vulnerabilities. If you remove all unnecessary components, you reduce security flaws.
- Many Docker users choose to use Alpine Linux, which is an extremely stripped down Linux distribution. You get the bare bones Linux.
    - Different variants of Linux: Alpine, Ubuntu, Debian, etc..
- Alpine is only 4-5 mb in size! Additionally, the runtime footprint is less.
- Due to the small size, this means deploying to a server is very fast with the small download size.
- Alpine is built to be secure. This does not however mean you should just assume it’s good enough. Security teams should still scan it for vulnerabilities.
- You’ll need to install any necessary software in your Dockerfile. You don’t even get curl.

Images - other small images

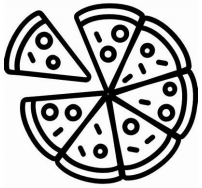
- Newer images now exist for small images that bring the same benefits of Alpine, but in other possibly more friendly Linux distros. They offer a bit more balance of size and functionality than Alpine.
  - Ubuntu Minimal 18.04 is 29 mb
  - Debian Slim is around 88 mb

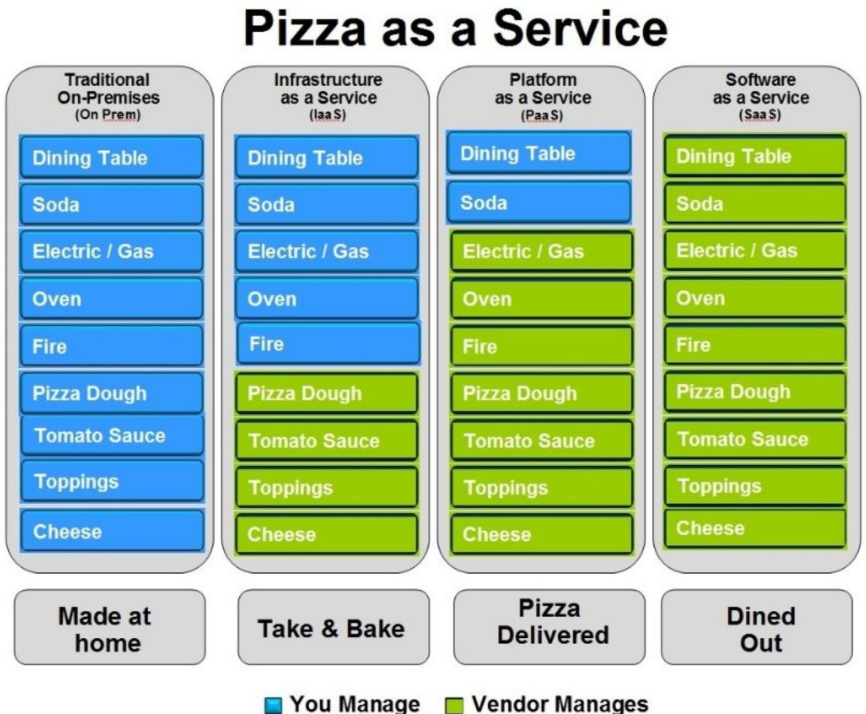
Kubernetes

DevelopIntelligence
A PLURALSIGHT COMPANY

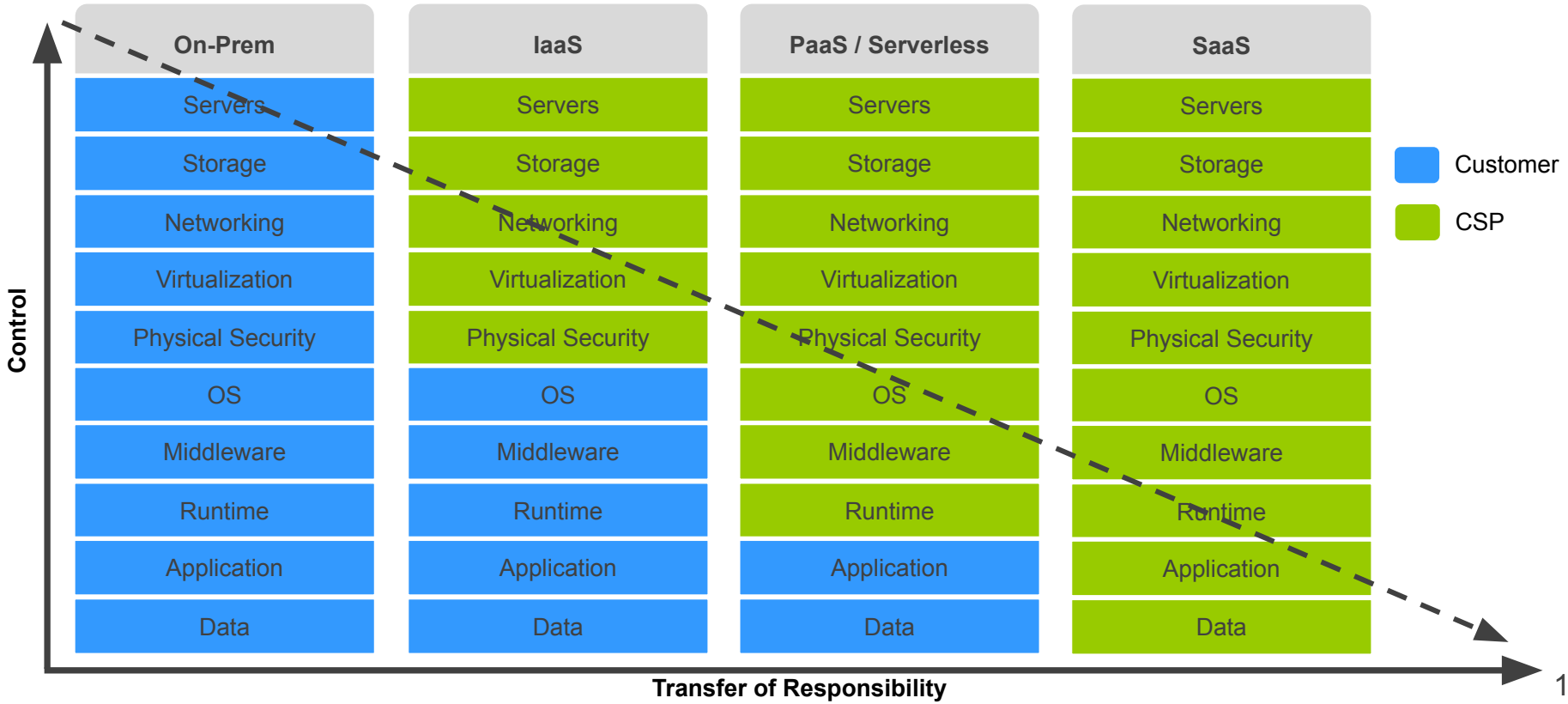
Pizza-as-a-Service

From a LinkedIn post by Albert Barron from IBM
(https://www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service/**)**





Side-by-Side Comparison



What is Kubernetes?

Container Orchestration. Derived from the Greek word for helmsman or pilot. Created by Google, open sourced in 2015

K8s makes containers at scale possible

K8s history

- 2003 - internal @ Google, they used a system called Borg to help with cluster management
- 2013 – Docker released, makes containers easier than they've ever been by providing a solid ecosystem – docker images
- 2015 – K8s version 1.0 is open sourced/released and also CNCF formed at same time
  - Some of CNCF founding members include Google, Twitter, Docker, VMWare, IBM..
  - Goal of CNCF to advance adoption of container technology
- Amazon's ECS released 6 days after K8s
  - ECS not open source like K8s
  - ECS – EC2 Container Service
  - Meant to work with other Amazon services

Container Orchestration – why?

Container Orchestration – why?

- Why do we need something that tells our containers what to do?
- When dealing with thousands of containers on hundreds of servers, there must be some orchestration

K8s terminology: pod, node, cluster

- Pod
  - Wrappers around your container
- Node
  - Collection of pods
  - Docker Desktop runs a single node K8s cluster
  - So, think of your laptop as a ‘node’ right now
- Cluster
  - Collection of Nodes
  - Analogy: server rack
- So, remember the hierarchy is your application -> Docker container -> pod->node->cluster
- Namespace
  - Ensures no collisions
  - Used for requesting capacity in production

Docker K8s vs miniKube

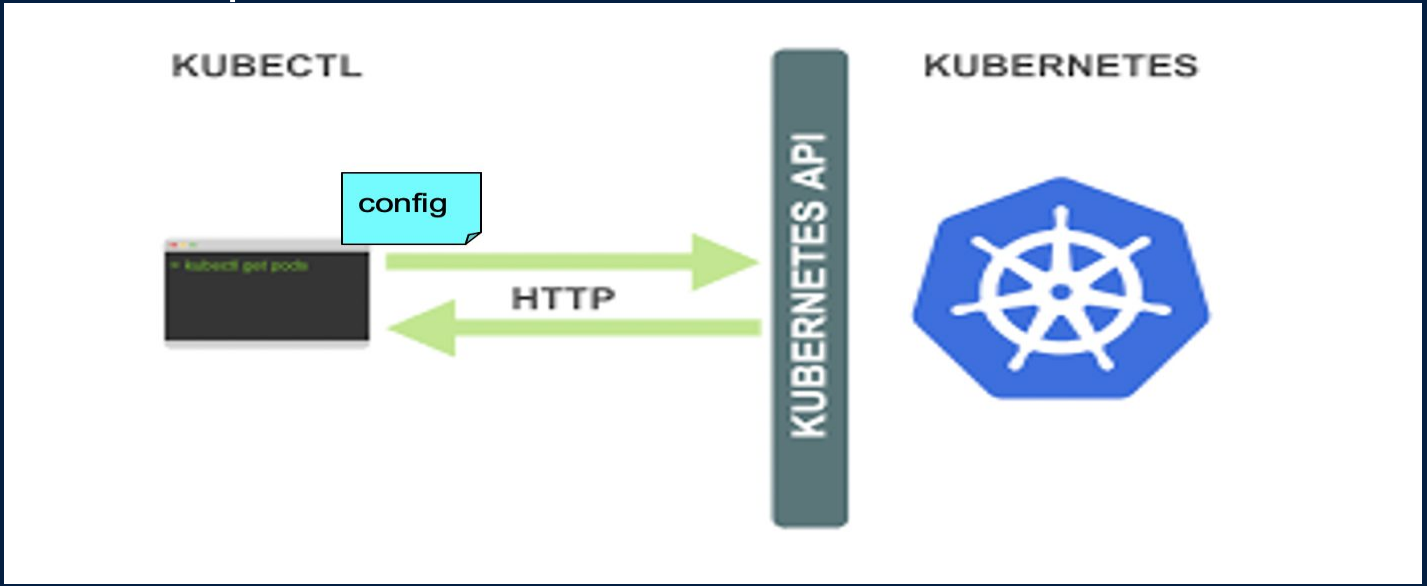
- Docker Desktop now comes with K8s (DD version 18.06 or higher on Mac)
  - We will use this to run K8s – you can see it in preferences…
  - It sets up a single node cluster locally – this is automatically set up for you!
- Prior to this minikube was the most popular software to run K8s locally
- Minikube sets up a separate VM, which meant that you would have to access any K8s services you built locally using that VM IP address
- But, DD uses Apple's Hypervisor framework so we can access services using localhost instead of the VM IP address (same idea on Windows)

A note on Docker Swarm

- Docker Swarm is also a container orchestrator
- But K8s far more popular and “enterprise-grade”
  - Even Docker somewhat conceded by including K8s within Docker Desktop

Kubectl

- CLI for running commands against K8 clusters
- K8s exposes an API that we can call into



KUBECTL commands

Common Kubectl Commands:

- Apply – apply a template file to your cluster, its like create/update
  - kubectl apply –f workloads.yaml
- Get <service, pod, etc> <name> - get information about the resource
  - kubectl get pod mypod
- Get all – return a full page of details about your cluster's resources
- Describe – get detailed information about a resource
  - kubectl describe service myservice
- Logs – return console output from a pod (or a container inside it)
  - kubectl logs mypod
- In production, you may also want to provide a namespace for any and all kubectl commands:
  - Kubectl –n <namespace> logs mypod

KUBECTL – connect to a pod

Common Kubectl Commands:

- Exec – launch a bash shell inside a pod
  - kubectl exec –it <pod-name> -- /bin/bash
- To connect to a service (we'll talk about services shortly):

  kubectl -n <namespace> get svc sample-service

KUBECTL contexts

Kubectl can be pointed to different K8s clusters for management. Since we activated Kubernetes for Docker, you'll need to tell kubectl which cluster to work on. You do this with context.

- kubectl config view – shows config information, including available contexts
- kubectl config current-context – shows the current context
- kubectl config use-context docker-desktop – switch kubectl to send commands to the cluster identified by the name "docker-desktop"

You can also switch contexts using the Docker menu by expanding the Kubernetes item. For instance, scus-lab-a1 is a context as that's the cluster we deploy to for Electrode!

KUBECTL example – get all

- Let's run kubectl get all, what do we see:

```
vn0silf@m-c02xr115jgh7 wcnp-deep-dive % kubectl get all
NAME              READY   STATUS    RESTARTS   AGE
pod/hello-pod     1/1     Running   8          26d
pod/nginx-pod     1/1     Running   14         61d

NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP        62d
service/nginx-service     NodePort    10.106.89.56    <none>        80:30080/TCP   60d
```

- It shows me both pods and services I currently have running
- Be sure to have Kubernetes context set correctly in Docker Desktop
  - You might see no context available, you will have to enable it in your preferences! Preferences -> K8s -> Enable K8s

K8s - pods

- Pods are lowest/smallest unit of work within K8s
- Containers at a high level are wrappers around your applications
- Pods are another wrapper around your containers
- Your application -> Docker container -> K8s pod
- Typically just one container per pod
- Pods are mortal
  - This is key, so remember this!

K8s – more on pods

- So, K8s doesn’t manage containers – it manages pods
- Every pod gets a unique IP address
- That IP address is not available outside the K8s cluster
  - So, by default, pods are not accessible outside the k8s cluster
  - In practical terms that means we can’t directly access the application running in our container
- For instance, if we are only running a pod with a Jenkins Docker image we can’t open our browser and access an endpoint with our running instance of Jenkins

Lab 1: KUBECTL + pod exercise – apply

1. First, ensure that you have Kubernetes up and running Docker Desktop!
2. Copy the contents from the pod.yaml file here locally in a new folder (just name the folder k8s): https://github.com/varoonsahgal/k8s/blob/main/pod.yaml

1. Use **kubectl apply -f pod.yaml** to apply that file – basically that creates a pod with the Docker image specified
2. Let's do a **kubectl get all or kubectl get pods** to view it
3. Use **kubectl describe** to get more info about the pod
4. Now, lets launch a shell inside our pod to play around:
    1. Run kubectl exec -it my-nginx-pod -- /bin/bash

Lab 1: KUBECTL + pod exercise – apply

5. Now, lets launch a shell inside our pod to play around:
    1. Run kubectl exec -it my-nginx-pod -- /bin/bash
    2. Once inside the pod, try running curl http://localhost
        1. But, it will complain that curl is not found!
        2. So, we can use apt – advanced package tool – which is used to install/remove software on various Linux distros
        3. Now, do **apt-get update**
        4. Then, run **apt-get install curl**
        5. Finally, try to **curl localhost** again
        6. You should get a response this time w/ the nginx default html!
        7. Type “exit” to leave the shell
        8. FROM nginx:1.17.0
        9. CMD apt-get update
        10. CMD apt-get install CURL
        11. docker build Dockerfile -> gives us a new image -> then we use that image in our pod.yaml

6. Question: If I delete the pod, do another apply to create a new one, will the curl library still be there inside the new pod?

7. Takeaway from lab: we don’t yet have a way of accessing the pod from our own browser – we haven’t exposed it through a service. That’s coming up!

KUBECTL example – describe

- With describe we can get a lot more information about our pods
  - Much like “inspect” in the Docker CLI
- Let’s run kubectl describe on our new pod: kubectl describe pod/my-nginx-pod
- Notice ClusterIP for the service - this is the IP address assigned to the pod available WITHIN the cluster not OUTSIDE of the cluster
  - In other words, you can’t use this IP address to connect to the cluster from outside world
- Pods can exist without services, but they will NOT be accessible, so let’s see how to access one
  - And a service without a pod is just an empty endpoint - will give you a “host not found”

KUBECTL delete

- Let's delete our pod
- **Kubectl delete -f pod.yaml**
  - We can delete it by specifying the file used to create it via apply
  - We can also delete by name: **kubectl delete pods/pod-name**
- Can we "stop" pods in K8s like we stopped containers in Docker?
  - Not really, as that concept doesn't transfer from Docker to K8s pods
  - Closest thing in K8s is to set replicas of a pod to 0 (we'll cover this soon)
- Go ahead and apply the pod again as we will need it for next lab – **kubectl apply –f pod.yaml**

K8s + Docker images

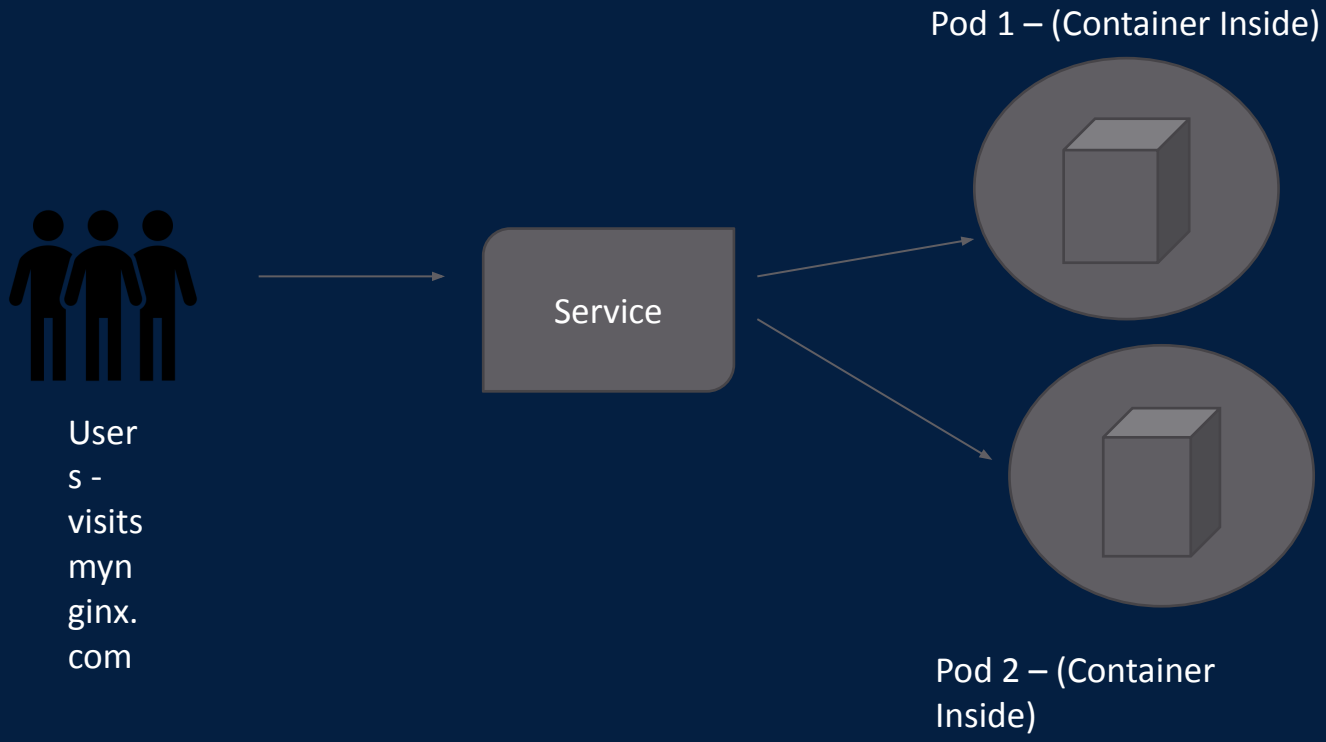
- Where does k8s look to find images?
  - It will first look at your machine to see if you have any images
  - If it doesn't find it there, then it will pull it from DockerHub
  - You can also specify the repo where it looks
- Remember we have this in our pod.yaml file (See containers section):

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx:1.17.0
```

K8s – services

- So, how do we expose our pods to users? Through something called a ***service*** in K8s
- Why services? Why not just access pods directly?
- Because pods are mortal
  - K8s often needs to kill pods – an app might be over consuming resources
  - Or when we are scaling up dramatically, new pods are added and removed at a fast pace, which makes having a stable endpoint even more important
- So, **services** give us a stable, single point of entry into our pods…

K8s – services: expose pods



K8s - services

- Like pods, services are another K8s construct
- Services are what essentially expose our pods – which hold our applications - to the outside world
  - So services are a gateway to our applications in K8s
- Now, if pod dies, service will handle the IP address mapping between the endpoint that the user hits and the new pod replacing it
- You can think of services as load balancers for your pods, which distribute traffic across pods

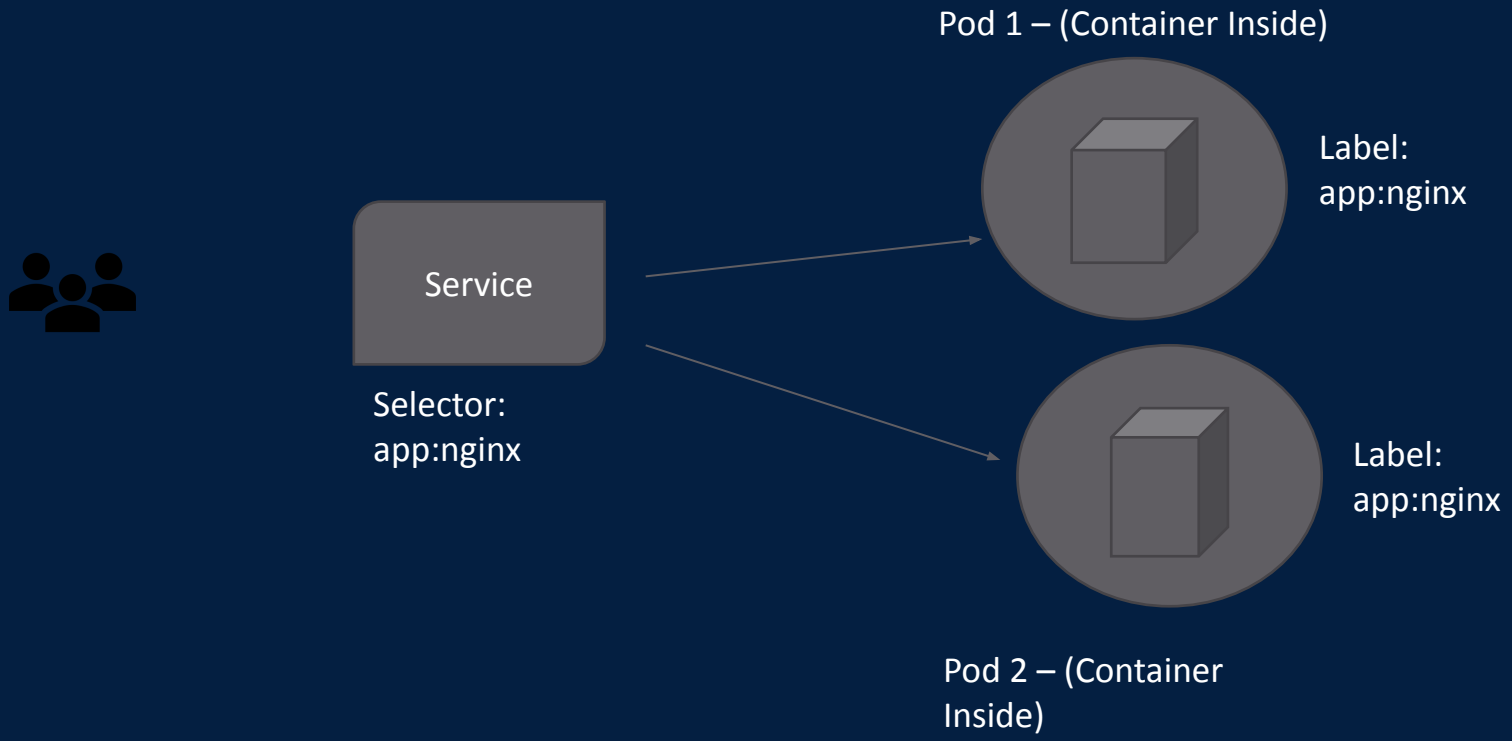
K8s – service types

- ClusterIP – default option. Provides access inside the cluster. Good for services that should remain internal to the application like a database.
- NodePort – Provides a port outside of the cluster. Makes your service public.
  - A ClusterIP Service, to which the NodePort Service routes, is also automatically created when you use a NodePort.
  - Note: the port you expose on the cluster is only valid from 30000-32767 –
  - **We will be using this service type in our lab to expose a pod that we create!**
- LoadBalancer – Exposes the Service externally using a cloud provider's (ie GKE) load balancer.
  - Only valid on cloud implementations, and creates a load balancer and attaches it to your service.

K8s labels– services + pods

- How do we tell a service to forward traffic to a pod?
- Through labels!
  - Labels are in key:value format
  - Pods can have multiple labels
- So, a service would have a label selector and then a given pod will have a label which would match that label selector
  - Which in turn means that the service will forward traffic to that particular pod!
- Basically, the service says “all pods on the cluster with these labels are mine and i will load balance them!”

K8s – labels in services + pods



K8s – Service.yaml

service.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    nodePort: 30001
  type: NodePort
```

- Notice the selector: **app:nginx** – in our pod.yaml we also defined a label saying **app:nginx**
  - Because the selector matches our pod.yaml label, K8s will know to put our pod behind this service!
  - NodePort is the port which we can use to access this service from browser
    - Remember port mapping in Docker? This is same idea except with K8s – nginx is running on port 80, and we are exposing it on 30001

Lab 2: K8s services

1. Our goal here is to apply a service locally and connect it to our nginx pod. That way, we can access nginx through our browser
    1. Remember – pods are not directly accessible outside of K8s cluster, we must expose them through a service!
2. Look at the service.yaml file here: https://github.com/varoonsahgal/k8s/blob/main/service.yaml
    1. Make a copy of that file locally in the same directory as your pod.yaml file
3. Now, apply the file to create the service in K8s! Run **kubectl apply –f service.yaml**

Lab 2: K8s services pt 2

4. You should now be able to view nginx running from your browser – go to http://localhost:30001
    - So, our service is exposing our pod, which in turn is running the nginx Docker container! Beautiful!
5. Try describing the service via kubectl to get more info – **kubectl describe service/my-nginx-service**
6. Try deleting the pod – **kubectl delete -f pod.yaml,** and now try accessing the service in the browser – you should see nothing!
7. Go ahead and reapply the pod

K8s – sidecar

- Typically you would only put one container per pod
- In rare cases, a pod can have multiple (>1) containers if those containers are tightly coupled
  - The sidecar pattern is one example of when we would have multiple containers in a single pod
  - In this pattern a secondary container's only job is to support the primary container
  - For example, you may have a secondary container that just sends logs from the primary container to a central logging system

K8s – declarative model

- Example of a pod.yaml file:

K8s – declarative model

- K8s uses infrastructure as code concepts
  - *Infrastructure as code (IaC) means managing your IT infrastructure using configuration files.*
- Declarative vs imperative:
  - Declarative is like asking someone to bake a cake and giving them the ingredients without a recipe – the baker will figure it out
  - Imperative is like giving ingredients + recipe - telling baker exactly what to do..ie here are the exact steps you need to take to bake a cake

K8s Replicasets

- What is self healing? When k8s automatically spins up a new pod when one goes down
- Replicasets and deployments are the two ways to achieve self-healing within K8s
- When creating a Replicaset, we don’t need to define the pods as a separate resource – ie as a separate pod.yaml file.
  - Pods are created within the replicaset and are “owned” by that replicaset

Lab - K8s Replicasets

- Grab the replicaset file and apply it locally: https://github.com/varoonsahgal/k8s/blob/main/replicaset.yaml
  - - **kubectl apply -f replicaset.yaml**
- You can see the pods created by the replicaset if you do a **kubectl get all**
- Try deleting a pod and see what happens – it should self heal and bring it back up (kubectl delete pod/pod-name)
- Look at desired, current and ready states!
- Your service should still be running
- Note that this replicaset is still attaching to our service through the label selector
- so if service is still running you should be able to see the nginx response in the browser by going to http://localhost:30001/
- Try deleting the service, do you still see the nginx homepage?
- Everything under template is specific to the pods its creating
  - Note that we don't explicitly use the word pod in the replicaset.yaml file, it's implied!

K8s Deployments – the what

- Deployments are another K8s construct (like pods, services, replicasets)
- Deployments actually create replicasets
- Why do we need deployments if they just use replicasets under the hood?
- Because deployments **allow us to rollback to prior application states** if something went wrong with our rollout
  - For instance the new version of our application does not work as planned, so we want to go back to the earlier working version. Very easy in K8s through deployments!
  - Plus we still get self healing because we are using replicaset(s) created

K8s - Triggering Deployments

- A new deployment will trigger a new replicaset to be created
- How do we trigger a new deployment rollout? By updating the pod template in the deployment.yaml file (like the labels or image)
  - We would then need to reapply the deployment.yaml as well
  - Note that if we update the number of replicas that does not trigger a “new” deployment, just an update to the number of pods
- If we have 1000’s of replicas, it will take longer to rollout the new updates –so some users may see the older version of your application post-deploy until all pods have been updated

K8s Deployments – Rolling updates

- If we go from v1 to v2 of our image (our application), then there there will be an intermediate phase where some pods are running on v1 and some on v2!
- This is what's known as a **rolling update**, and ensures that we don't experience any downtime!
- For our lab exercise, let's pretend we are "upgrading" our application from an Apache server to a Nginx server – so we will modify the Docker image being used in the deployment that we will create….
  - So, we will go from Apache -> Nginx

K8s – Kubectl Deployment commands

- You can rollback with: **kubectl rollout undo deployment apache-deployment**
- You can see rollout status with: **kubectl rollout status apache-deployment**
- You can see rollout history with: **kubectl rollout history deployment apache-deployment**
- You can pause and resume rollouts
- More details in the docs: `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`

Lab - K8s Deployments

- Delete any replicasets/pods that you might have running via **kubectl delete** (but leave the service)
- Grab the deployment file and apply it locally: https://github.com/varoonsahgal/k8s/blob/main/deployment.yaml
  - - **kubectl apply -f deployment.yaml**
- Do a **kubectl get all** and notice that it creates a replicaset, 10 pods, and also a “deployment”
- Try deleting a pod and see what happens – it should self heal and bring it back up
- Look at desired, current and ready states!
- Your service should still be running
- Note that the **deployment** is still attaching to our service through the label selector – we still need a **service** to expose our pods!
- If service is still running you should be able to see the Apache response in the browser by going to http://localhost:30001/

Lab - K8s Deployments – pt 2

- Now, let's update our deployment.yaml file to use nginx image instead of httpd inside the deployment.yaml file. Go ahead and reapply the deployment file via **kubectl apply –f deployment.yaml**
- **This is the equivalent of doing a new release for our application!**
- Now, notice that a new replicaset has been created and the old one still remains
  - The old one is there to allow us to roll back if we need to!
- Go to http://localhost:30001/ and you should see that it's updated to use the nginx homepage!
- Notice that you never experienced any downtime – because K8s is gradually terminating the old Apache pods and replacing them with the new nginx pods – it's not killing all 10 pods at once and then bringing up 10 new ones
- Also, notice that the old replicaset is still maintained – this is key for rollbacks!
- So, what if we want to rollback to our earlier deployment with the Apache container image?
  - Simple!
- Just run **kubectl rollout undo deployment apache-deployment**

GCP - GKE, Google Kubernetes Engine

DevelopIntelligence

A PLURALSIGHT COMPANY

GKE - creating a cluster

When creating a new cluster in GKE, you will notice 2 options:

1. Standard mode
    a. You define the # of nodes you need - ie 3 nodes, 10 node, etc..
2. Autopilot mode
    a. This is relatively new to GCP
    b. GOAL: reduce operating cost in running K8s clusters
    c. hands-off, an abstraction layer that does the dirty work essentially
    d. analogous to Fargate in AWS ECS/EKS

Let's go ahead and create a K8s cluster so that we can play with it! Run through this lab:

https://github.com/varoonsahgal/gcp-training/wiki/Creating-a-cluster-in-GCP

Google Cloud shell - a quick note

Note that we will be doing our work in Google cloud shell - which will provide us command line access to the cloud resources hosted on GCP. This way, we can execute commands against the k8s resources we create through GKE

K8s - HPA

DevelopIntelligence

A PLURALSIGHT COMPANY

Autoscaling - what and WHY

- Why do we even want to enable autoscaling?
  - variable workloads - sometimes our apps are busier
- why would we NOT want to enable autoscaling?
  - predictable workloads - not much variation in demand

HPA - horizontal pod autoscaler - the HOW

- If we want auto-scaling in K8s, how do we get it?
- we would need the HPA
- This is another resource in K8s like services, pods, etc.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

What are the options for autoscaling?

- Command line - kubectl scale deployment
- GKE Console
- Manifest file (best option - give us IaC)
- Load testing - see this: https://github.com/varoonsahgal/gcp-training/wiki/Generating-load-to-test-our-autoscaling

Autoscaling in command line

To scale directly from the command line you can run this command (this will create an HPA resource):

kubectl autoscale deployment hello-world-rest-api --cpu-percent=50 --min=1 --max=10

Autoscaling in GKE console

To autoscale a deployment in GKE using the console, you can follow these steps:

Go here: https://github.com/varoonsahgal/gcp-training/wiki/Scaling-in-GKE

1. Navigate to the GKE cluster that you want to configure autoscaling for.
2. Click on the Workloads tab, then select the deployment that you want to autoscale.
3. In the deployment details view, click on the Horizontal Pod Autoscaling tab.
4. Click the "Enable autoscaling" button.
5. Configure the autoscaling settings, including the minimum and maximum number of replicas, target CPU utilization, and target memory utilization. You can also configure other advanced options, such as custom metrics.
6. Click the "Save" button to enable the autoscaling configuration.

Once autoscaling is enabled, Kubernetes will automatically adjust the number of replicas based on the current resource usage and the configured target utilization levels. You can monitor the autoscaling behavior and resource usage in the GKE console or through Kubernetes tools such as kubectl.

It's important to note that autoscaling is not always necessary or appropriate for all workloads, and should be carefully configured based on your specific requirements and resource constraints.

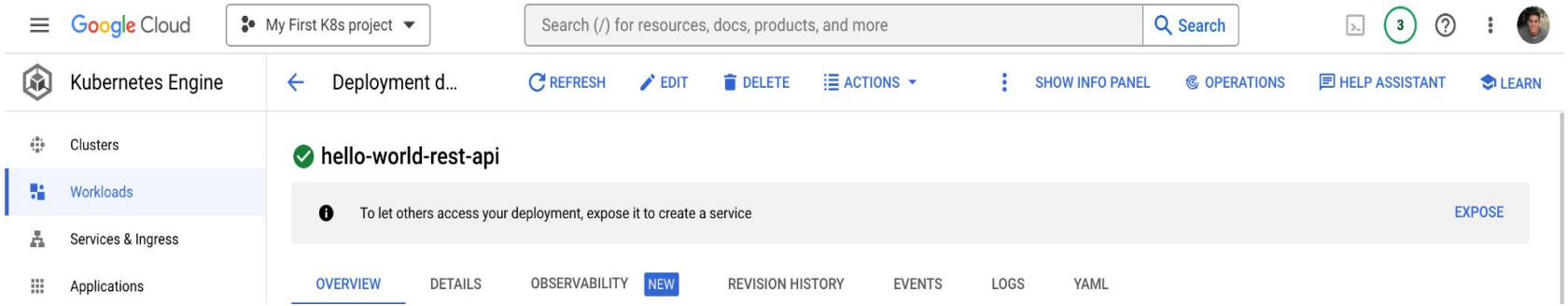
Cluster auto-scaling

- in addition to scaling pods, we can set up scaling of nodes - as seen here:
  - https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler#autoscaling_limits

K8s Services

Services in GKE

- Note that we can expose our deployments with a service and in fact are prompted to do so:



Deployments in K8s

Deployments before K8s

- Before DevOps, cloud and Docker orchestration came along, production support and releases was a wild west full of danger and long nights.

- Releases were a MAJOR nightmare for many. Spending most/all of a night on the phone with an entire ops team supporting a release and fixing issues was commonplace. The build pipeline was slow and prone to errors, and frequently took days to complete. When releases failed, you “failed forward” instead of rolling back.

Deployments overview

- Deployments are an abstraction that creates ReplicaSets and manages pods being launched into different ones in a controlled way.
- They are declarative and you provide the end state. Kubernetes takes care of getting your pods to that end state in a managed way, without downtime.
- On a rollout, K8S creates a new replicaset and starts moving pods to the new one in a controlled way, before removing the old replicaset.
- Note: deployments create a replicaset programmatically. You do not manage the replicaset directly!
- You can see rollout status with: kubectl rollout status deployment nginx-deployment
- You can see rollout history with: kubectl rollout history deployment nginx-deployment
- You can rollback with: kubectl rollout undo deployment nginx-deployment
- You can pause and resume rollouts
- There are tons of settings here: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Deployments vs Replicasets

- A ReplicaSet ensures that a specified number of pod replicas are running at any given time. However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.

- Therefore, it’s recommended to use Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.

- This actually means that you may never need to manipulate ReplicaSet objects: use a Deployment instead, and define your application in the spec section.
- This is straight from the k8s docs: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#when-to-use-a-replicaset

Deployments - Rollouts

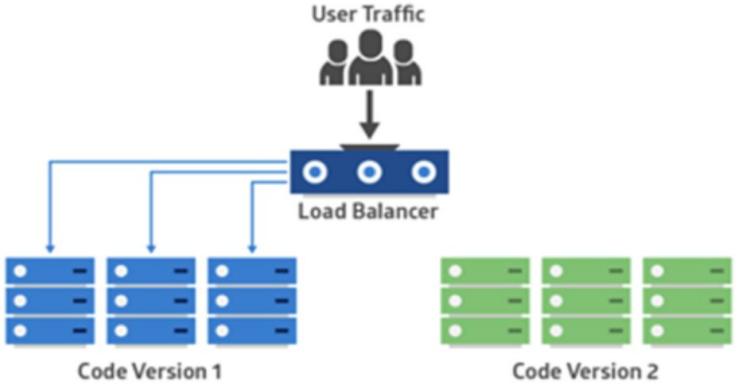
- When you update the pod definition in your Deployment, the DeploymentController will start a rolling update process. It does this by simply managing ReplicaSets for you.
    - Assume you already have code running in your Deployment.
- A new ReplicaSet is created with the new Pod configuration. The Replicas count is zero.
- The Replicas count will be increased on the new ReplicaSet.
- Once the pods are launched, the Replicas count on the original ReplicaSet are reduced.
- This process will continue until the new ReplicaSet has the original Replicas count and the old ReplicaSet has a Replicas count of zero.
- The old ReplicaSet will hang around empty as a record of past deployments. By default 10 empty replicasets hangaround before being garbage collected (this is customizable in your Deployment spec). A rollback will reverse this process.
- A Deployment's rollout is triggered if and only if the Deployment's Pod template is changed, for example if the labels or container images of the template are updated. Other updates, such as scaling the Deployment, do not trigger a rollout.
- If you have 1000s of services, it will take a long time and user will see different versions based on request allocation.

K8s - deployment methods

- Kubernetes supports rolling deployments, which is default. But sometimes applications can’t work with that type of deployment and need a full cutover. You need another option.
- Blue/green is a common pattern, where an entire new copy of the application is created, traffic is drained to the old deployment, and then switched over all at once.
- We need blue/green because of breaking changes like database or API changes where its not possible to keep both versions alive.
- Blue/green requires double the resources.
- Blue/green is not supported by K8S. But you can still implement it yourself manually or with some scripting.
- You can also do canary deployments in the same way as blue/green, but by creating a small Deployment of the new code, using common labels in your Service so that the canary pods are mixed in with original pods. Once the code is vetted, then the canary Deployment can be expanded and the original Deployment reduced.

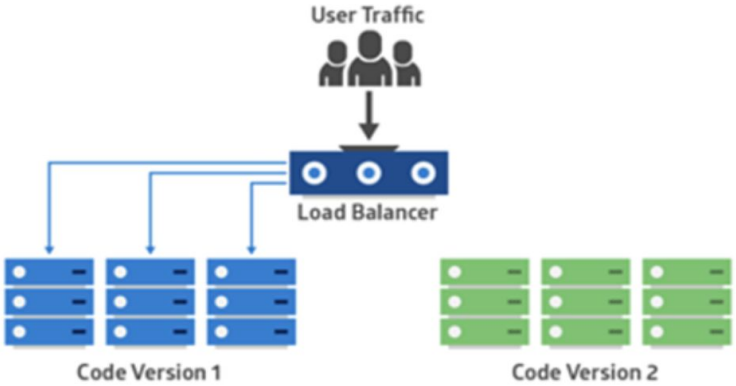
K8s - Blue/Green deployments

- Some applications can not have multiple versions running simultaneously, which would happen in an upgrade using a ***rolling*** upgrade
- A Blue-Green deployment strategy, also known as Red-Black Deployment in software delivery, is one in which the old and new instances of an application or microservice operate in parallel in production in the same time with a load balancer switching traffic from the older version to the newer one.
- Blue == the currently running version of the application, green is the new version
- Once green is ready, ALL traffic is re-routed to the green version
- This way we get a zero-downtime upgrade



K8s - Blue/Green deployments - why?

- We need blue/green because of breaking changes like database or API changes where its not possible to keep both versions alive.
- Sometimes it's OK to have 2 different versions up and running for a short time but in those cases we would use a rolling update not a blue-green!



RBAC in k8s/GKE

DevelopIntelligence

A PLURALSIGHT COMPANY

RBAC in K8s

- Who can access resources in my GKE cluster?
  - Who can touch this pod?

Where do users in K8s live?

- K8s does NOT maintain a database of users

K8s - Kubeconfig file

- Kubernetes uses a YAML file called kubeconfig to store cluster authentication information for kubectl
- By default in GKE, the file is stored at $HOME/.kube/config
- We can override that file when we try to do something in kubectl - we will do that to “fake” a user trying to access resources
  - This is to demo how RBAC can work - once we assign that user a role they will be able to access the resources…

k8s - PRINCIPLE OF LEAST PRIVILEGE

- When adding verbs to the role, we should be sure to use least privilege principle:

```
rules:
- apiGroups: ["example.com"]
  resources: ["*"]
  verbs: ["*"]
```

Caution: Using wildcards in resource and verb entries could result in overly permissive access being granted to sensitive resources. For instance, if a new resource type is added, or a new subresource is added, or a new custom verb is checked, the wildcard entry automatically grants access, which may be undesirable. The principle of least privilege should be employed, using specific resources and verbs to ensure only the permissions required for the workload to function correctly are applied.

RBAC is additive

- RBAC allows just the addition of privileges - ie you can not DENY access to certain K8s resources
- For that reason you should choose the IAM role with the least privileges and add the required privileges via RBAC on top.

IAM vs RBAC

- What is IAM - identity and access management
  - ie - which users in GCP can access what resources?
  - We typically have many users in GCP - we don't want everyone to be able to do anything in GCP
  - So IAM dictates who can do what…
  - let's walkthrough an example -
- You can use both IAM and RBAC to control clusters in K8s
- READ more here: https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control
-

IAM vs RBAC

- Note that IAM is not a feature of K8s - rather it's a feature of the cloud provider, in our case GCP
- K8s RBAC ***is*** a feature of K8s
- https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control
-

K8s RBAC hands on lab

- Let’s see how to use RBAC in k8s
- Suppose we have to give developers the ability to read from a cluster - but not the ability to modify anything inside our K8s cluster
- How can we do that?
- Well, we would use RBAC in k8s!

K8s users

- In K8s, there is no concept of “users”
- Users would be created by GCP IAM,
  - so we can create different users with access to our k8s cluster via GCP IAM
  - but then were would k8s RBAC come into play??
  - you do not have to create Google Account/GCP users to create Kubernetes users. Users and groups in Kubernetes are very lightweight "objects" and are derived directly from the authentication system in use

K8s Configmaps and Secrets

DevelopIntelligence

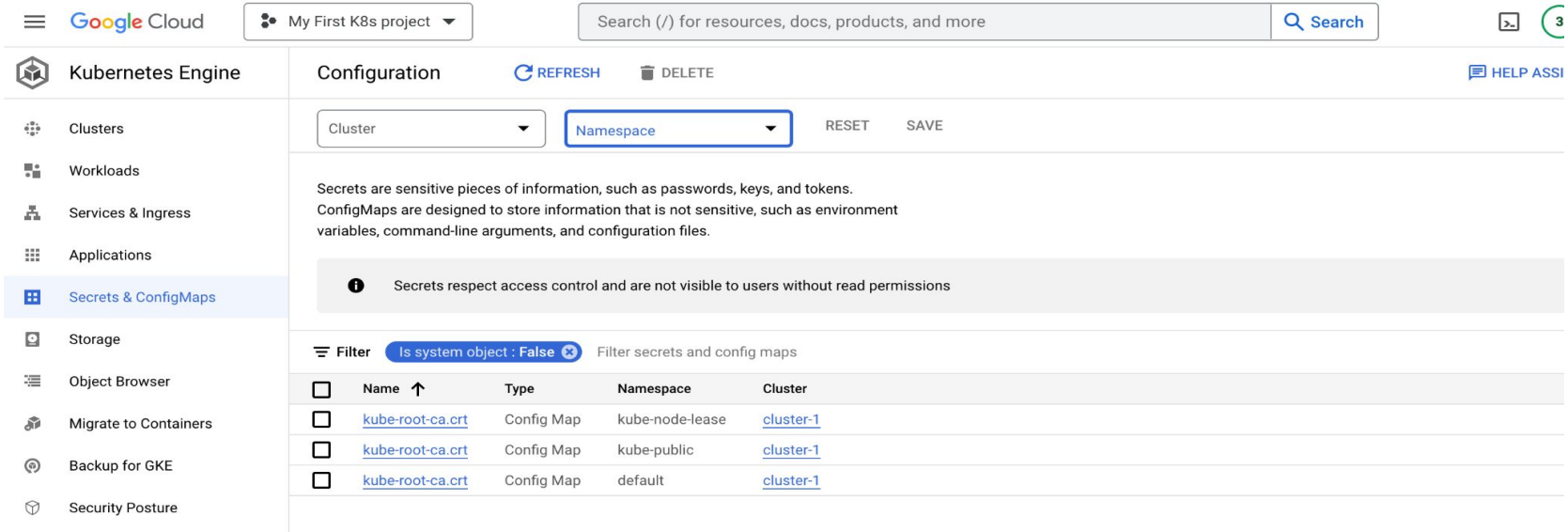
A PLURALSIGHT COMPANY

K8s - Configmaps

- Containers often need several environment variables to function properly so that configuration is externalized.
  - But passing in variables directly to containers is messy.
- ConfigMaps allow you to decouple configurations at the cluster level and have containers refer to them.
  - Configmaps should only store NON-sensitive config data that can be used by an applications

Configmaps

- Here we can see the option to create secrets and configmaps inside of Kubernetes engine - these are some default configmaps used internally by K8s:



K8s - Configmaps

- Configmaps allow us to reuse configurations.
- ConfigMap is centralized and when updated it immediately updates the changes in the variables.
- Configmap is not displayed by kubectl get all, maybe to avoid clutter.
- kubectl get cm or kubectl get configmap will display it.

Configmaps vs secrets

- When would we use configmaps vs secrets?
- Configmaps should be used for **non-secret** configuration data - like environment variables for endpoints
- Secrets should be used for things that are actually secret and need to be hidden - like database credentials, API keys, etc..

K8s - volumes

- The filesystem in a container is ephemeral, because containers are immutable. Volumes provide persistent storage on the host system.
- Storage can be tricky, we will only discuss it at a high level.
  - Secrets use volumes so you should at least know what they are at a high level - details are not necessary right now
- We want things to be stateless but volumes provide attached persistence
  - I personally rarely use volumes but instead remote persistence like databases, s3 bucket etc.
- For some off the shelf software like jenkins or wordpress you need volumes, it is a necessity for lift and shift projects as well but you can do it differently for greenfield project.
- Volumes can be a variety of types, from the host hard drive to cloud storage volumes like AWS EBS, Local, NFS, Cider etc.
- To read up on the details: https://kubernetes.io/docs/concepts/storage/volumes/
- Kubernetes presents idea of volumes through abstractions where you can plugin different kind types of storage using provisioner which understands implementation of volume type.

K8s - secrets

- 🤔 How do you store sensitive information?
  - Should you include it in a Docker image?

K8s - secrets

- Secrets are small pieces of sensitive information that your pods can access at runtime. Think passwords, SSH keys, etc.
- Secrets are stored as volumes that your containers can access. Alternatively, they can be exposed as environment variables.

K8s - secrets

- When using kubectl get, you won't see the contents of a secret.
- Note that secrets are still accessible to those with access directly to the cluster i.e. view in git or exec to echo env variable.
- Secrets are meant to protect from including them in Docker images which are more portable.
    - It is a ***best practice*** to have secrets managed by a limited set of people who know how to keep them safe.
        - And of course - don't just check them into source control alongside your resources.
- When secrets are updated, the containers automatically pick up the changes immediately.

K8s - secrets - how are they accessed

- How do we access secrets once they are created?
- Typically 2 ways in the k8s pod:
  - 1. Environment variables
  - 2. Mounted in the container file system

K8s secrets encoded not encrypted

- What’s the difference between encoding and encryption?

Why do we encode data?

- base64 is encoding not encryption
- it's pretty easy to decode something that's been encoded - simply go to https://www.base64decode.org/ or a similar website to encode/decode!

- so why do we even encode in the first place? answer: data obfuscation
  - ***encoding can provide some level of security by obfuscating the data in a way that's not readable by humans***
  - obfuscate == make sensitive data less readable by humans
- However, it is important to note that base64 encoding is not a form of encryption and does not provide any additional security beyond obfuscation.
- So - it is not a substitute for strong encryption and other security measures.

Why do we encrypt data?

- base64 is encoding not encryption
- it's pretty easy to decode something that's been encoded - simply go to https://www.base64decode.org/ or a similar website to encode/decode!

- so why do we even encode in the first place? answer: data obfuscation
  - ***encoding can provide some level of security by obfuscating the data in a way that's not readable by humans***
  - obfuscate == make sensitive data less readable by humans
-

K8s - secrets exercise part 1

- Let’s create a secret via the command line, go ahead and execute this command:
- Let’s see how to create a secret in our GCP cluster as well!

K8s - secret manifest file

- Secrets have a ***type***
- The Secret type is used to facilitate programmatic handling of the Secret data
- The values for all keys in the data field have to be base64-encoded strings.
- Opaque is the default secret type
  - it's for arbitrary-user defined data
- More details on types here:
  - https://kubernetes.io/docs/concepts/configuration/secret/#secret-types
  -

K8s - secrets exercise

- Exercise file: https://github.com/varoonsahgal/k8s/blob/main/secret.yaml
- Review the file, notice values are base64 encoded and go ahead and do a kubectl apply to your GKE cluster
- Also describe the MongoDB pod and notice the output references the secret. Also notice that it was mounted as a volume to the container for you.
- List all secrets **kubectl get secret** (note that kubectl get all will not display the secret)
- Describe the secret kubectl describe secret mongosecret
- Exec into the pod kubectl exec -it <pod> -- /bin/bash and echo the variables
  - echo $MONGO_INITDB_ROOT_USERNAME
  - echo $MONGO_INITDB_ROOT_PASSWORD
- When done, delete the resources

Note: for simplicity, the secret and deployment are together. Do not do this in a

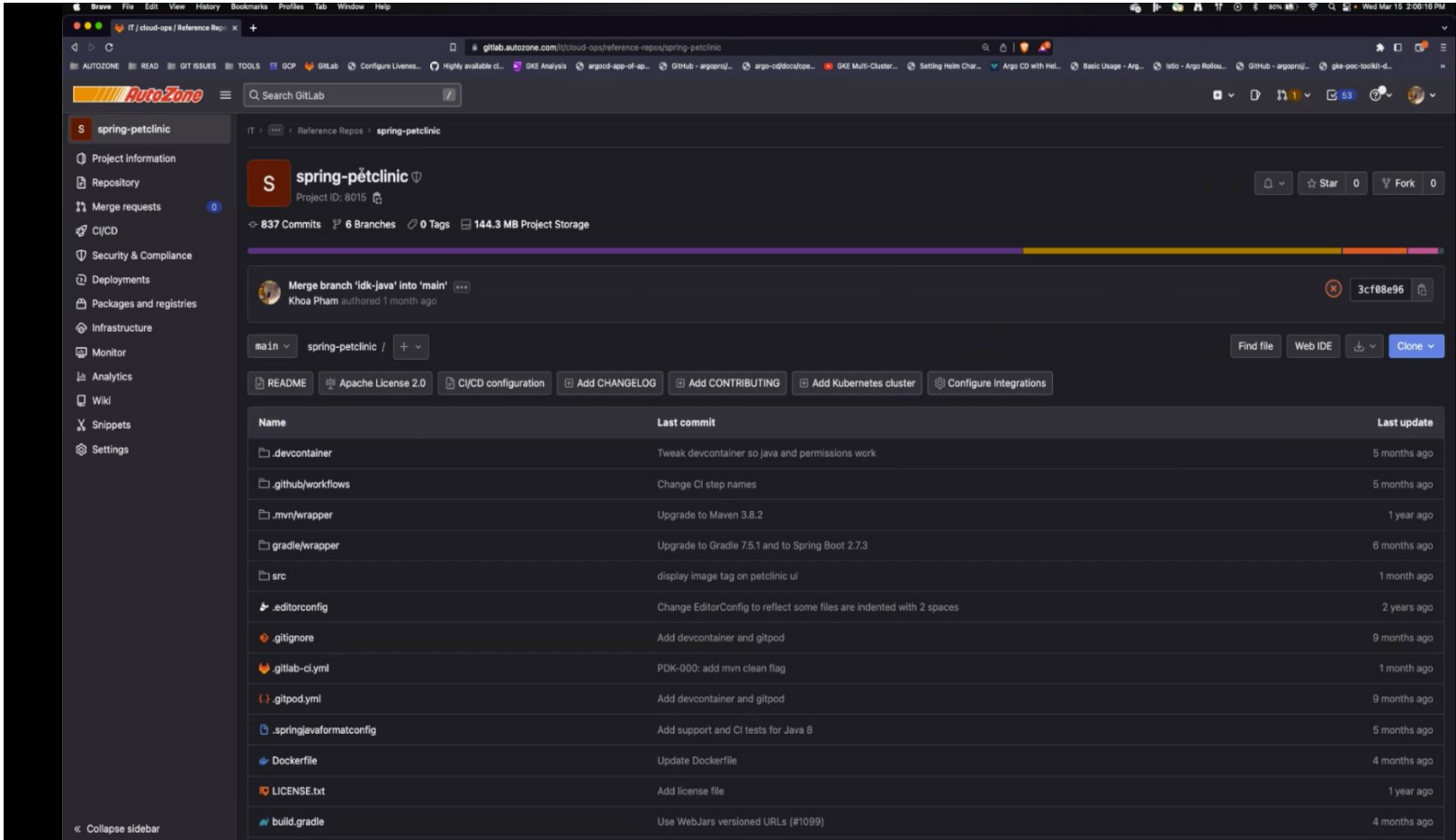
Secrets - best practices

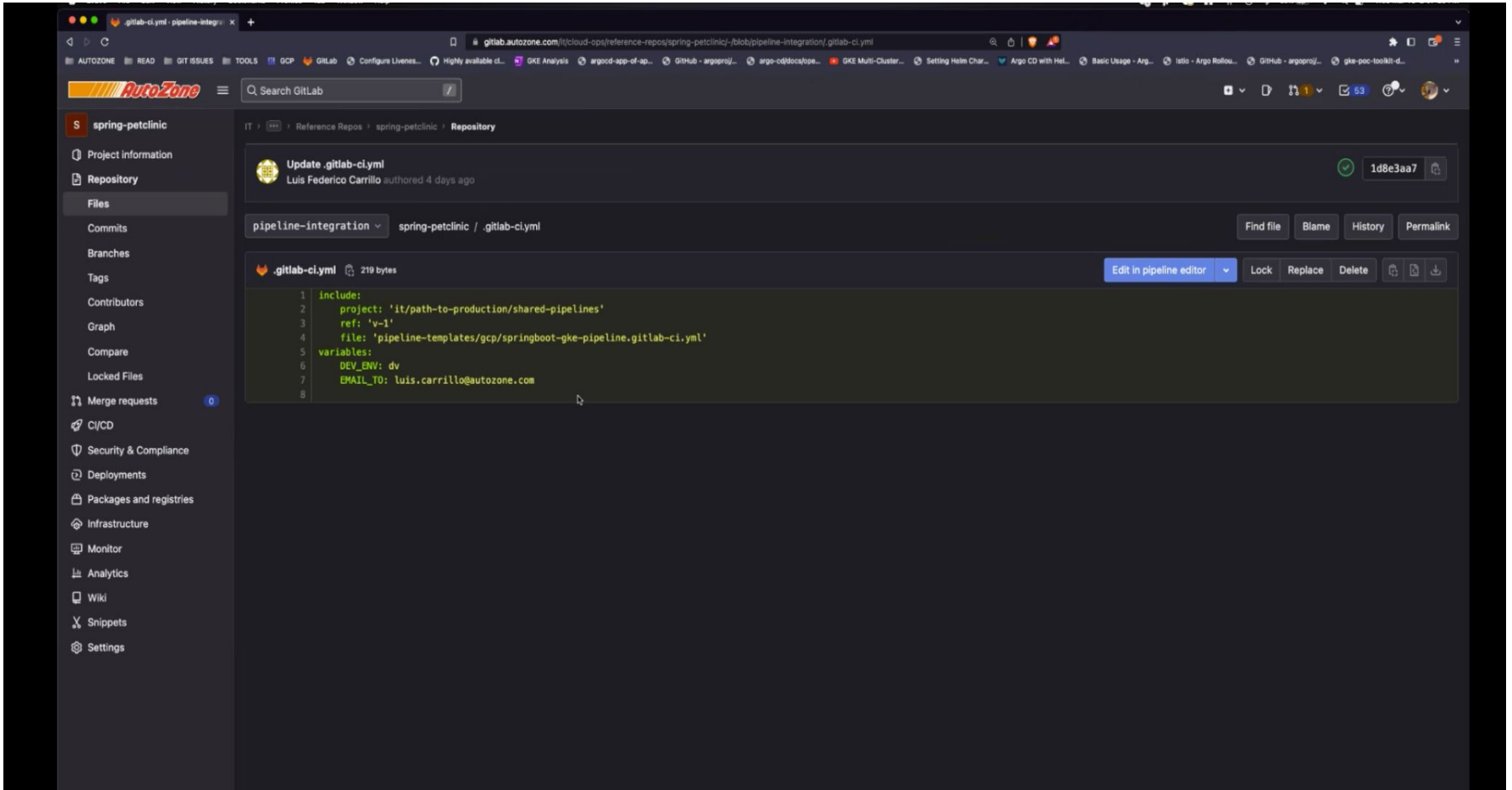
- By default secrets are unencrypted - anyone with access to your kubernetes API can retrieve or modify a secret
- For that reason it's good practice to take these steps to safely use secrets: 👍
- Enable Encryption at Rest for Secrets.
- Enable or configure RBAC rules with least-privilege access to Secrets.
- Restrict Secret access to specific containers.
- Consider using external Secret store providers.
- Per the docs here: https://kubernetes.io/docs/concepts/configuration/secret/#:~:text=Kubernetes%20Secrets%20are%2C%20by%20default,anyone%20with%20access%20to%20etcd.

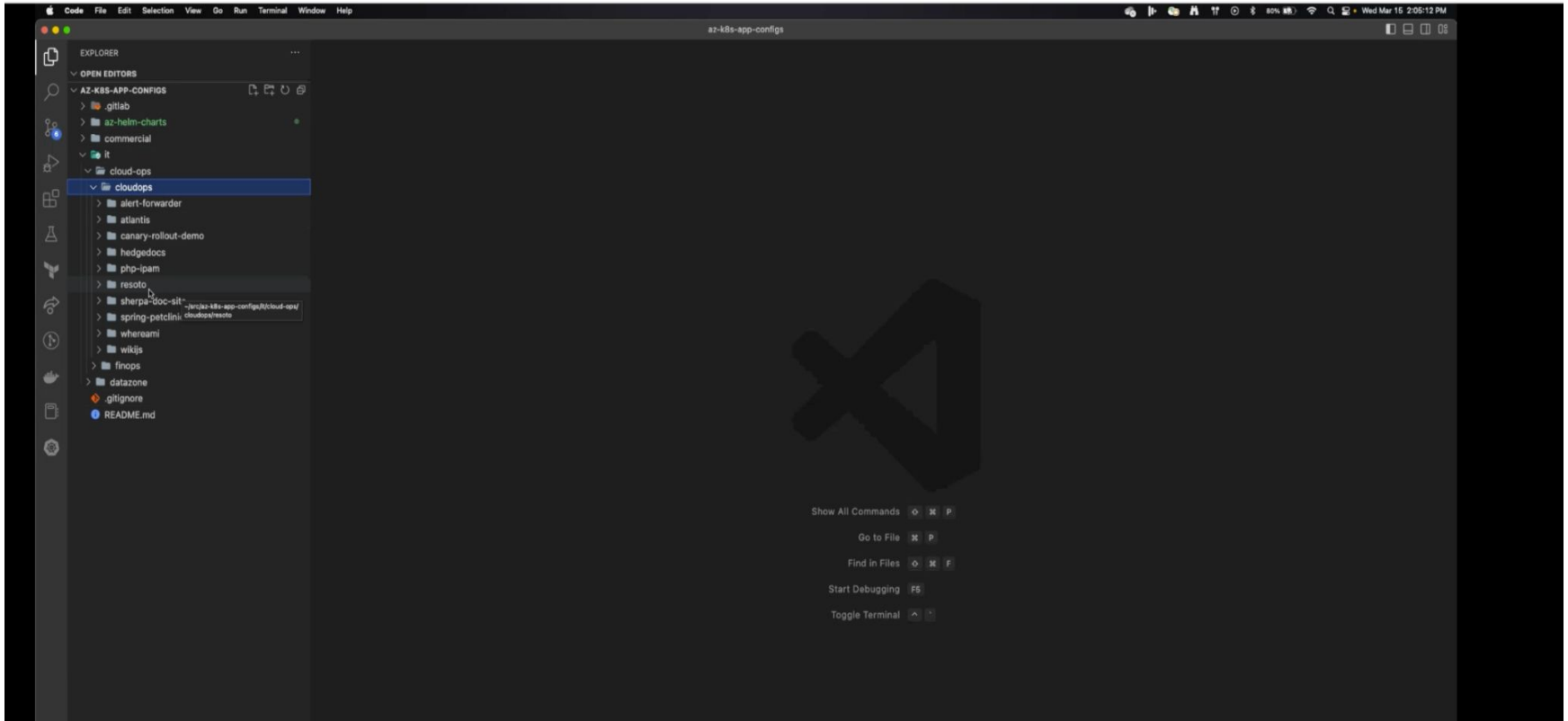
Secrets - exercise

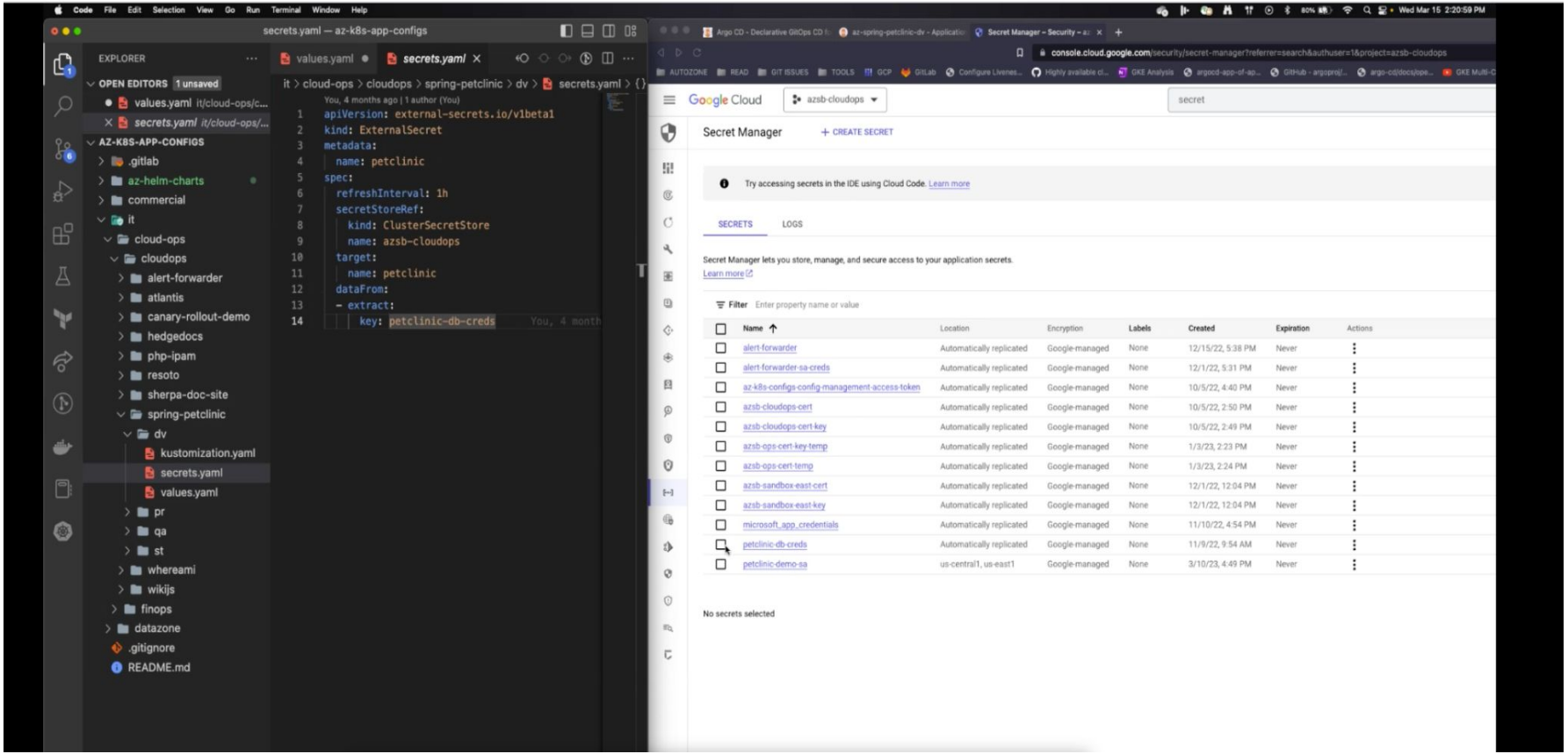
- Let’s get some practice creating secrets as a best practice in GKE

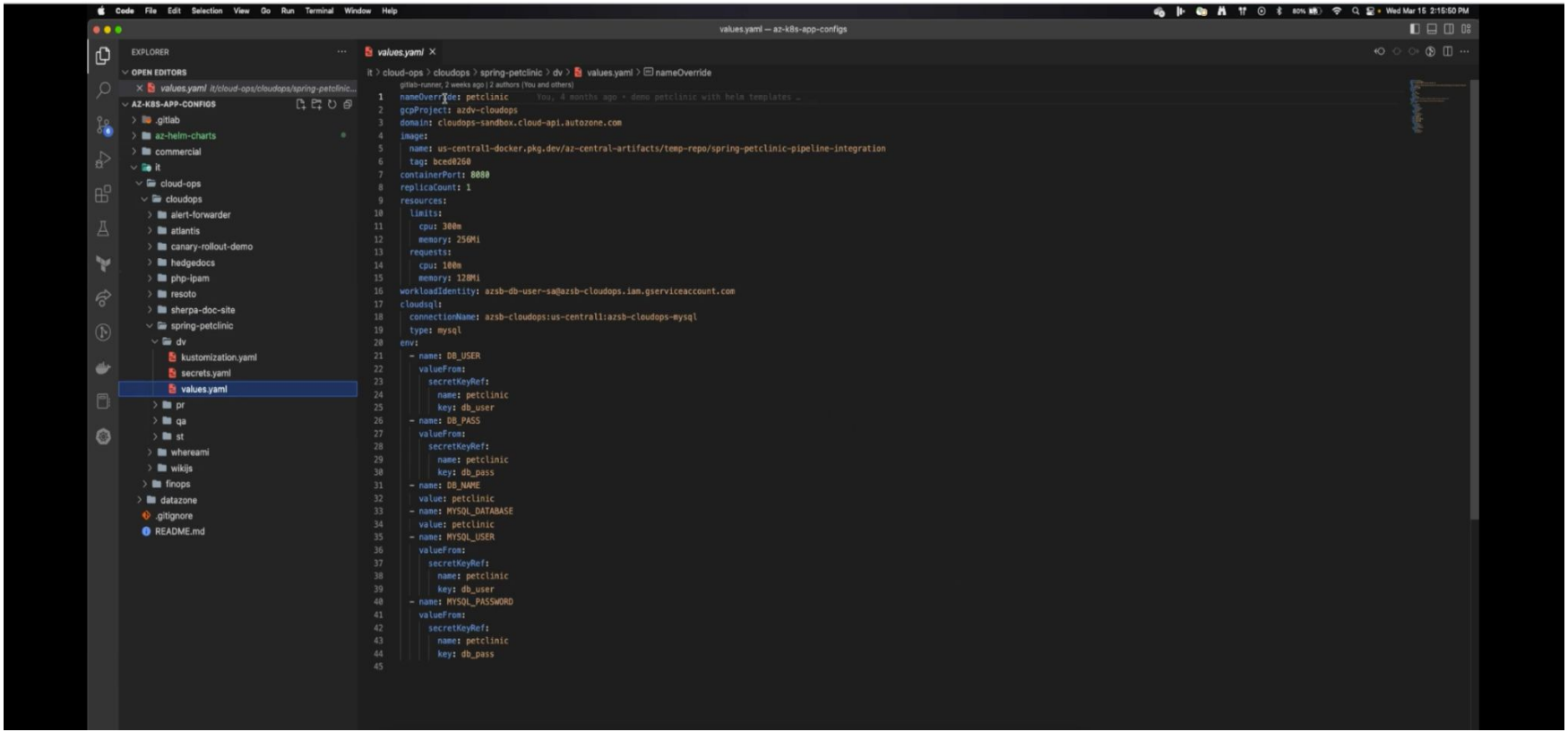
Autozone pipeline walkthrough

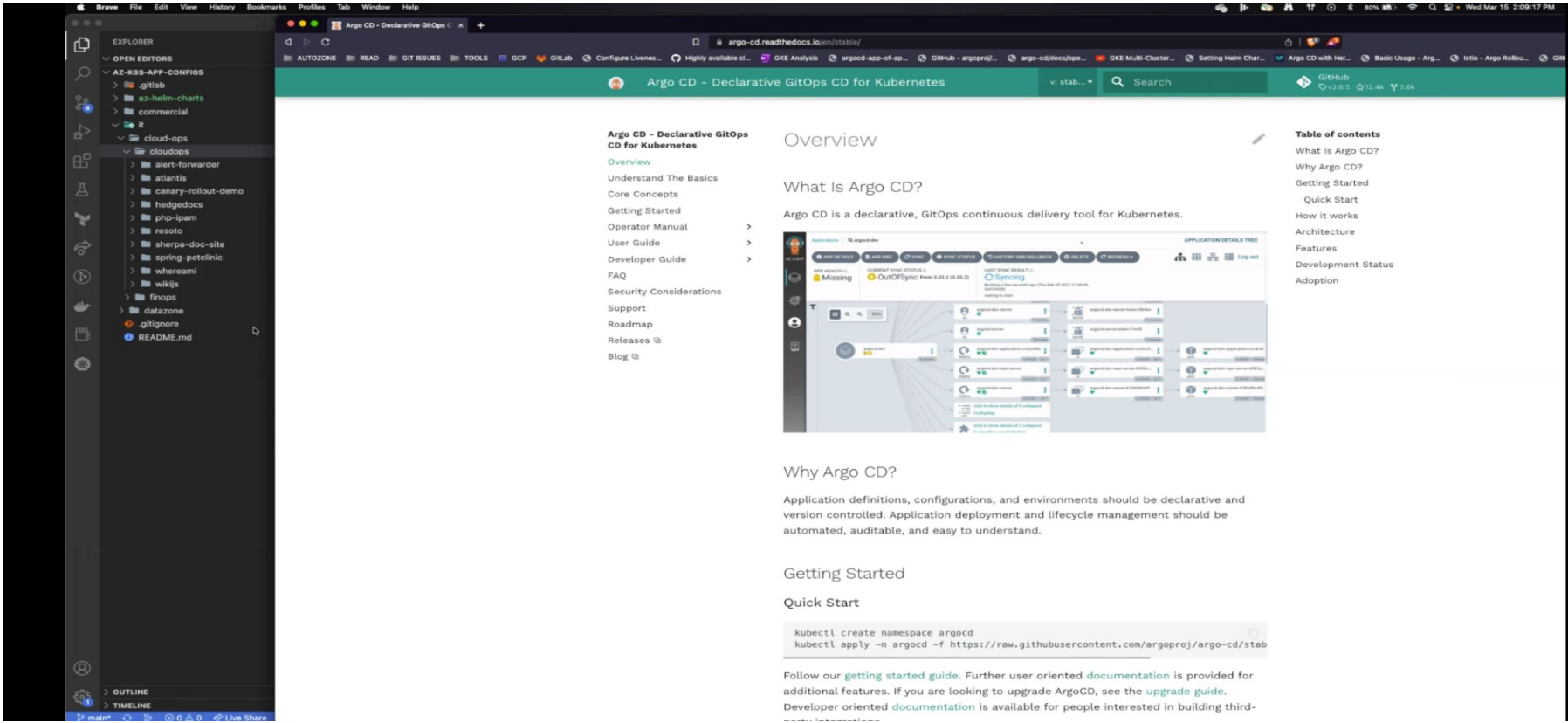

AutoZone
Search GitLab
spring-petclinic
Project information
Repository
Merge requests
CI/CD
Security & Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets
Settings
Collapse sidebar
IT › Reference Repos › spring-petclinic
spring-petclinic
Project ID: 8015
Star 0
Fork 0
837 Commits
6 Branches
0 Tags
144.3 MB Project Storage
Merge branch 'idk-java' into 'main'
Khoa Pham authored 1 month ago
3cf08e96
main
spring-petclinic /
Find file
Web IDE
Clone
README
Apache License 2.0
CI/CD configuration
Add CHANGELOG
Add CONTRIBUTING
Add Kubernetes cluster
Configure Integrations
Name
Last commit
Last update
.devcontainer
Tweak devcontainer so java and permissions work
5 months ago
.github/workflows
Change CI step names
5 months ago
.mvn/wrapper
Upgrade to Maven 3.8.2
1 year ago
gradle/wrapper
Upgrade to Gradle 7.5.1 and to Spring Boot 2.7.3
6 months ago
src
display image tag on petclinic ui
1 month ago
.editorconfig
Change EditorConfig to reflect some files are indented with 2 spaces
2 years ago
.gitignore
Add devcontainer and gitpod
9 months ago
.gitlab-ci.yml
PDK-000: add mvn clean flag
1 month ago
.gitpod.yml
Add devcontainer and gitpod
9 months ago
.springjavaformatconfig
Add support and CI tests for Java 8
5 months ago
Dockerfile
Update Dockerfile
4 months ago
LICENSE.txt
Add license file
1 year ago
build.gradle
Use WebJars versioned URLs (#1099)
4 months ago

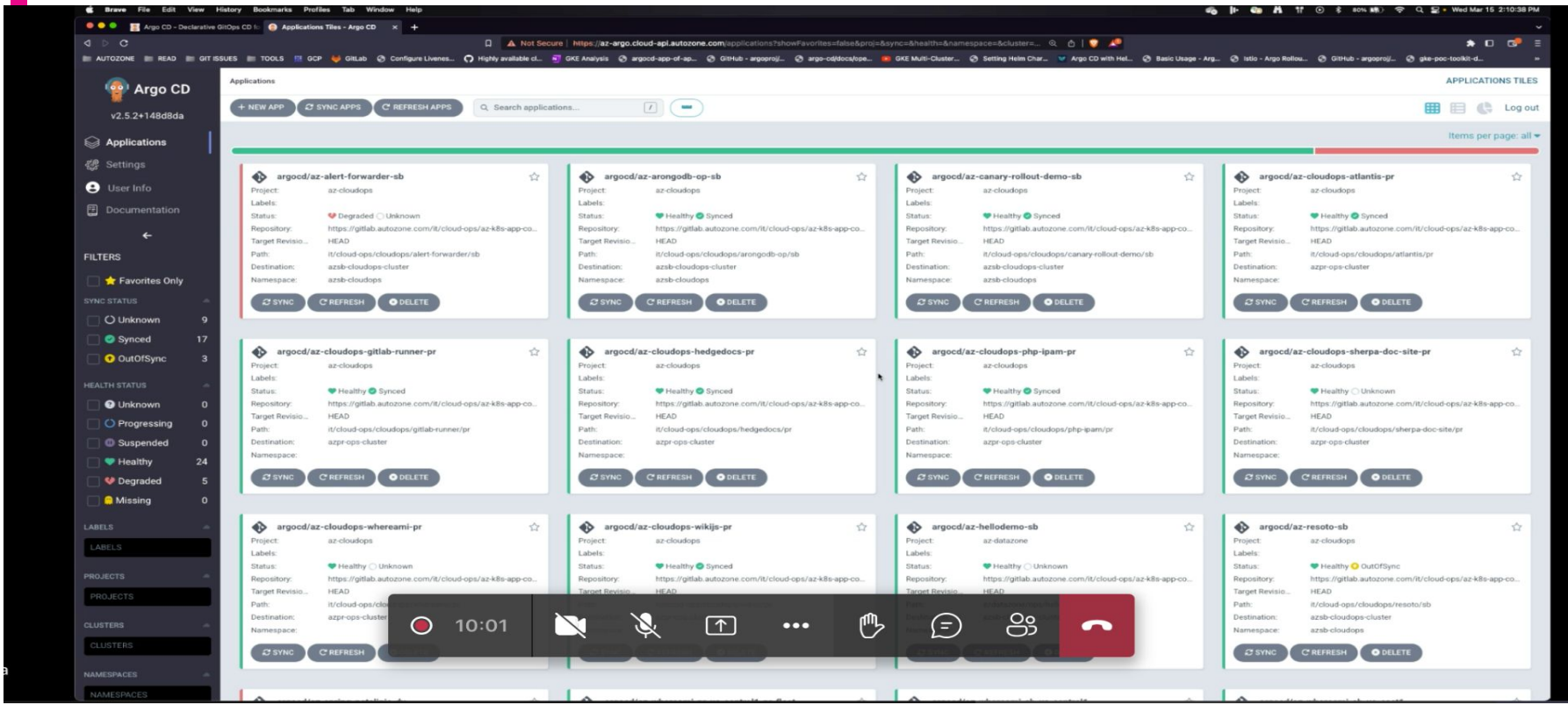

gitlab.autozone.com/it/cloud-ops/reference-repos/spring-petclinic/-/blob/pipeline-integration/.gitlab-ci.yml
AutoZone
Search GitLab
spring-petclinic
Project information
Repository
Files
Commits
Branches
Tags
Contributors
Graph
Compare
Locked Files
Merge requests
CI/CD
Security & Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets
Settings
IT › Reference Repos › spring-petclinic › Repository
Update .gitlab-ci.yml
Luis Federico Carrillo authored 4 days ago
1d8e3aa7
pipeline-integration
spring-petclinic / .gitlab-ci.yml
Find file
Blame
History
Permalink
.gitlab-ci.yml
219 bytes
Edit in pipeline editor
Lock
Replace
Delete
include:
project: 'it/path-to-production/shared-pipelines'
ref: 'v-1'
file: 'pipeline-templates/gcp/springboot-gke-pipeline.gitlab-ci.yml'
variables:
DEV_ENV: dv
EMAIL_TO: luis.carrillo@autozone.com


Code File Edit Selection View Go Run Terminal Window Help
Wed Mar 15 2:05:12 PM
az-k8s-app-configs
EXPLORER
OPEN EDITORS
AZ-K8S-APP-CONFIGS
.gitlab
az-helm-charts
commercial
it
cloud-ops
cloudops
alert-forwarder
atlantis
canary-rollout-demo
hedgedocs
php-ipam
resoto
sherpa-bioc-sit
spring-petclini
whereami
wikijs
finops
datazone
.gitignore
README.md
Show All Commands
Go to File
Find in Files
Start Debugging
Toggle Terminal


secrets.yaml — az-k8s-app-configs
EXPLORER
OPEN EDITORS 1 unsaved
values.yaml it/cloud-ops/c...
secrets.yaml it/cloud-ops/...
AZ-K8S-APP-CONFIGS
.gitlab
az-helm-charts
commercial
it
cloud-ops
cloudops
alert-forwarder
atlantis
canary-rollout-demo
hedgedocs
php-ipam
resoto
sherpa-doc-site
spring-petclinic
dv
kustomization.yaml
secrets.yaml
values.yaml
pr
qa
st
whereami
wikijs
finops
datazone
.gitignore
README.md
values.yaml
secrets.yaml
it > cloud-ops > cloudops > spring-petclinic > dv > secrets.yaml
You, 4 months ago | 1 author (You)
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
name: petclinic
spec:
refreshInterval: 1h
secretStoreRef:
kind: ClusterSecretStore
name: azsb-cloudops
target:
name: petclinic
dataFrom:
- extract:
key: petclinic-db-creds
Google Cloud
azsb-cloudops
secret
Secret Manager
+ CREATE SECRET
Try accessing secrets in the IDE using Cloud Code. Learn more
SECRETS
LOGS
Secret Manager lets you store, manage, and secure access to your application secrets.
Learn more
Filter
Enter property name or value
Name
Location
Encryption
Labels
Created
Expiration
Actions
alert-forwarder
alert-forwarder-sa-creds
az-k8s-configs-config-management-access-token
azsb-cloudops-cert
azsb-cloudops-cert-key
azsb-ops-cert-key-temp
azsb-ops-cert-temp
azsb-sandbox-east-cert
azsb-sandbox-east-key
microsoft_app_credentials
petclinic-db-creds
petclinic-demo-sa
No secrets selected


values.yaml — az-k8s-app-configs
EXPLORER
OPEN EDITORS
values.yaml it/cloud-ops/cloudops/spring-petclinic...
AZ-K8S-APP-CONFIGS
.gitlab
az-helm-charts
commercial
it
cloud-ops
cloudops
alert-forwarder
atlantis
canary-rollout-demo
hedgedocs
php-ipam
resoto
sherpa-doc-site
spring-petclinic
dv
kustomization.yaml
secrets.yaml
values.yaml
pr
qa
st
whereami
wikijs
finops
datazone
.gitignore
README.md
values.yaml
it > cloud-ops > cloudops > spring-petclinic > dv > values.yaml > nameOverride
gitlab-runner, 2 weeks ago | 2 authors (You and others)
nameOverride: petclinic
gcpProject: azdv-cloudops
domain: cloudops-sandbox.cloud-api.autozone.com
image:
name: us-central1-docker.pkg.dev/az-central-artifacts/temp-repo/spring-petclinic-pipeline-integration
tag: bced0260
containerPort: 8080
replicaCount: 1
resources:
limits:
cpu: 300m
memory: 256Mi
requests:
cpu: 100m
memory: 128Mi
workloadIdentity: azsb-db-user-sa@azsb-cloudops.iam.gserviceaccount.com
cloudsql:
connectionName: azsb-cloudops:us-central1:azsb-cloudops-mysql
type: mysql
env:
- name: DB_USER
valueFrom:
secretKeyRef:
name: petclinic
key: db_user
- name: DB_PASS
valueFrom:
secretKeyRef:
name: petclinic
key: db_pass
- name: DB_NAME
value: petclinic
- name: MYSQL_DATABASE
value: petclinic
- name: MYSQL_USER
valueFrom:
secretKeyRef:
name: petclinic
key: db_user
- name: MYSQL_PASSWORD
valueFrom:
secretKeyRef:
name: petclinic
key: db_pass


EXPLORER
OPEN EDITORS
AZ-K8S-APP-CONFIGS
.gitlab
az-helm-charts
commercial
it
cloud-ops
cloudops
alert-forwarder
atlantis
canary-rollout-demo
hedgedocs
php-ipam
resoto
sherpa-doc-site
spring-petclinic
whereami
wikijs
finops
datazone
.gitignore
README.md
OUTLINE
TIMELINE
argo-cd.readthedocs.io/en/stable/
Argo CD - Declarative GitOps CD for Kubernetes
Search
GitHub
Argo CD - Declarative GitOps CD for Kubernetes
Overview
Understand The Basics
Core Concepts
Getting Started
Operator Manual
User Guide
Developer Guide
FAQ
Security Considerations
Support
Roadmap
Releases
Blog
Overview
What Is Argo CD?
Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes.
Why Argo CD?
Application definitions, configurations, and environments should be declarative and version controlled. Application deployment and lifecycle management should be automated, auditable, and easy to understand.
Getting Started
Quick Start
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stab
Follow our getting started guide. Further user oriented documentation is provided for additional features. If you are looking to upgrade ArgoCD, see the upgrade guide. Developer oriented documentation is available for people interested in building third-
Table of contents
What Is Argo CD?
Why Argo CD?
Getting Started
Quick Start
How it works
Architecture
Features
Development Status
Adoption


Argo CD
v2.5.2+148d8da
Applications
Settings
User Info
Documentation
FILTERS
Favorites Only
SYNC STATUS
Unknown 9
Synced 17
OutOfSync 3
HEALTH STATUS
Unknown 0
Progressing 0
Suspended 0
Healthy 24
Degraded 5
Missing 0
LABELS
PROJECTS
CLUSTERS
NAMESPACES
Applications
APPLICATIONS TILES
+ NEW APP
SYNC APPS
REFRESH APPS
Search applications...
Log out
Items per page: all
argocd/az-alert-forwarder-sb
argocd/az-arongodb-op-sb
argocd/az-canary-rollout-demo-sb
argocd/az-cloudops-atlantis-pr
argocd/az-cloudops-gitlab-runner-pr
argocd/az-cloudops-hedgedocs-pr
argocd/az-cloudops-php-ipam-pr
argocd/az-cloudops-sherpa-doc-site-pr
argocd/az-cloudops-whereami-pr
argocd/az-cloudops-wikijs-pr
argocd/az-hellodemo-sb
argocd/az-resoto-sb
SYNC
REFRESH
DELETE
10:01

Google Cloud Functions

Google Cloud Functions - overview

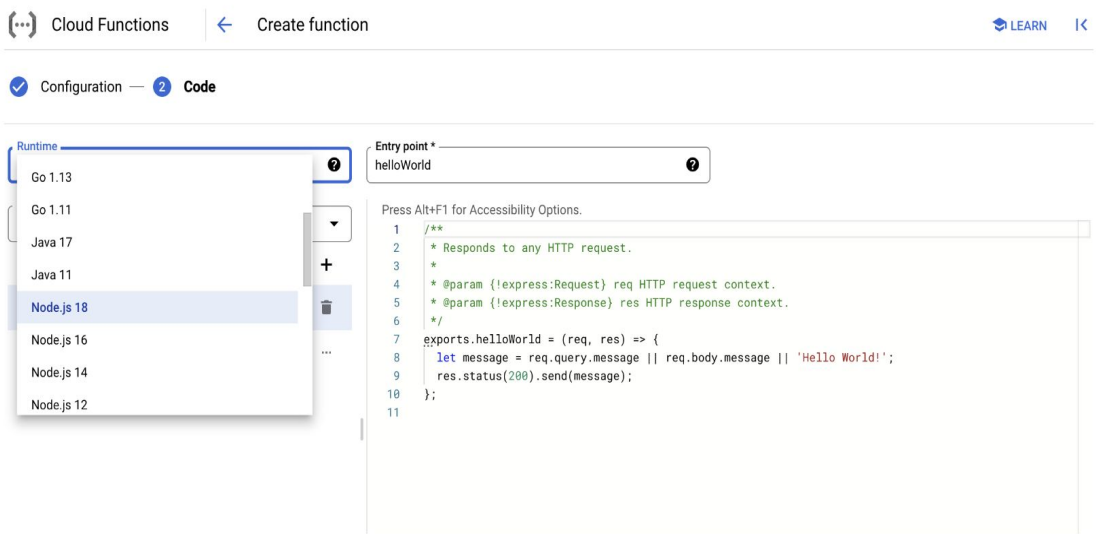
- Suppose you want to execute some code when some event happens
  - Suppose an image is added to Cloud Storage
    - What if we also want to create some metadata in a database for that image - like date added, image size, etc - and we wanted to do that automatically
    - In that case we could call a Cloud Function to add that image metadata to a database like Cloud SQL Mysql database
- Cloud Functions in GCP are like Lambdas in AWS

Google Cloud Functions - event driven

- Suppose you want to execute some code when some event happens
  - Suppose an image is added to Cloud Storage
    - What if we also want to create some metadata in a database for that image - like date added, image size, etc - and we wanted to do that automatically
    - In that case we could call a Cloud Function to add that image metadata to a database like Cloud SQL Mysql database

Cloud Functions - many programming languages

- You have many programming languages to choose from to write your functions:



1st gen vs 2nd gen

- Max timeout and max memory for cloud functions in 2nd gen is much higher than 1st gen
- Quick demo of this…

Pay for usage price model

- Price is based on a few factors:
  - # of invocations
  - Compute time of invocations (how resource intensive is your function?)
  - Memory and CPU provisioned
- Pricing per month below - generous free tier - after that its $.40 per million invocations a month

Invocations per month	Price/million
First 2 million	Free
Beyond 2 million	$0.40

Sample HTTP Cloud Function - Node.js


```
/**
 * Responds to any HTTP request.
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
exports.helloWorld = (req, res) => {
  let message = req.query.message || req.body.message || 'Hello World!';
  res.status(200).send(message);
};

```

Cloud functions - horizontal scaling

- Cloud functions scale horizontally
- There are some default autoscaling settings that you can override:
  - This means the more events, the more instances of your functions!

Cloud functions - more

- Cloud functions are recommended for event-driven use-cases
- They are NOT ideal for long running processes
  - This is b/c they are time-bound - they will timeout after a certain time
    - default is 1 minute, max is 60 minutes
- Cloud functions are basically serverless
  - Servers are still used behind the scenes by GCP
  - But, you don’t have to interact with any servers directly yourself

Thank you!

If you have additional questions, please reach out to me at:
(email address)

DevelopIntelligence
A PLURALSIGHT COMPANY