Welcome

Working with Kubernetes

DevelopIntelligence

A PLURALSIGHT COMPANY

Hello

About me...

- 27 years in the industry
- 23 years in teaching
- Certified Cloud architect
- Passionate about learning
- Also, passionate about Reese's Cups!

Prerequisites

This course assumes:

- You have a working knowledge of 1 or more programming languages
- This course is designed for existing software developers seeking to learn more about Kubernetes and how to use it

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

Agenda

- Kubernetes and container orchestration
- How do Docker and K8s fit together?
- Diving deeper into several Kubernetes primitives including Pods, Deployments, Services, ConfigMaps, Secrets, etc.
- Discussion of key areas of focus like configuration, observability, and persistence
- The architecture of a k8s cluster
- High-level overview of Helm

How we’re going to work together

- Slides / lecture
- Demos
- Team discussions
- Labs

Containerization as a Deployment Strategy

Application Hosting

By Application Hosting, we mean the target infrastructure and runtime platform used for deployment and execution of an application or system; can include compute (CPU and server resources), storage, network, data and operating system

Application Hosting – An “Interesting” Example?

Here’s an example of someone thinking “outside-of-the-box” when it comes to application hosting!

https://mashable.com/article/pregnancy-test-doom/

What Are the Hosting Options with Cloud?

- IaaS
- PaaS
- Serverless / FaaS
- SaaS
- Containers

What do they all mean?

Infrastructure-as-a-Service (IaaS)

- Involves the building out (and management) of virtual instances of:
  - Compute
  - Network
  - Storage

- Akin to spinning up a server (physical or virtual) in your location or data center complete with disks and required network connectivity

Infrastructure-as-a-Service (IaaS)

- The difference is in the where – instead of in your data center, it is created in a data center managed by one of the public Cloud providers

- Your organization is responsible for patching the OS, ensuring all appropriate security updates are applied and that the right controls are in place to govern interaction between this set of components and other infrastructure

Platform-as-a-Service (PaaS)

- Involves leveraging managed services from a public Cloud provider
- With this model, an enterprise can focus on management of their application and data vs. focusing on management of the underlying infrastructure
- Patching and security of the infrastructure used to back the managed services falls to the CSP (Cloud Service Provider)

Platform-as-a-Service (PaaS)

- Many managed services support automatic scale up or down depending on demand to help ensure sufficient capacity is in place
- Can be considered synonymous with the term “Cloud native”

Serverless / Functions-as-a-Service (FaaS)

- Also represents a type of managed service provided by the CSP
- Cost structure is usually consumption-based (i.e., you only pay for what you use)
- Supports many different coding paradigms (C#/.NET, NodeJS, Python, etc.)

Serverless / Functions-as-a-Service (FaaS)

- Typically, with Serverless (and PaaS), the consumer is only concerned with the application code and data – elements of the CSP's "backbone" used to support are managed by the CSP

- Includes more sophisticated automated scaling capabilities – built for Internet scale

Software-as-a-Service (SaaS)

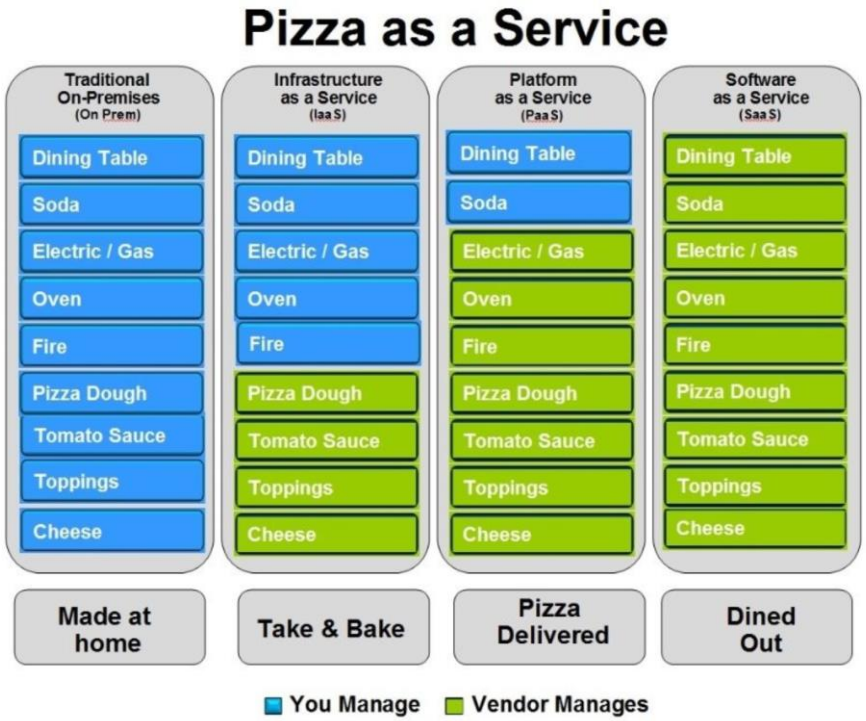
- Subscription-based application services
- Licensed for utilization over the Internet / online rather than for download and install on a server or client machine
- Fully-hosted and fully-managed by a 3rd party

Software-as-a-Service (SaaS)

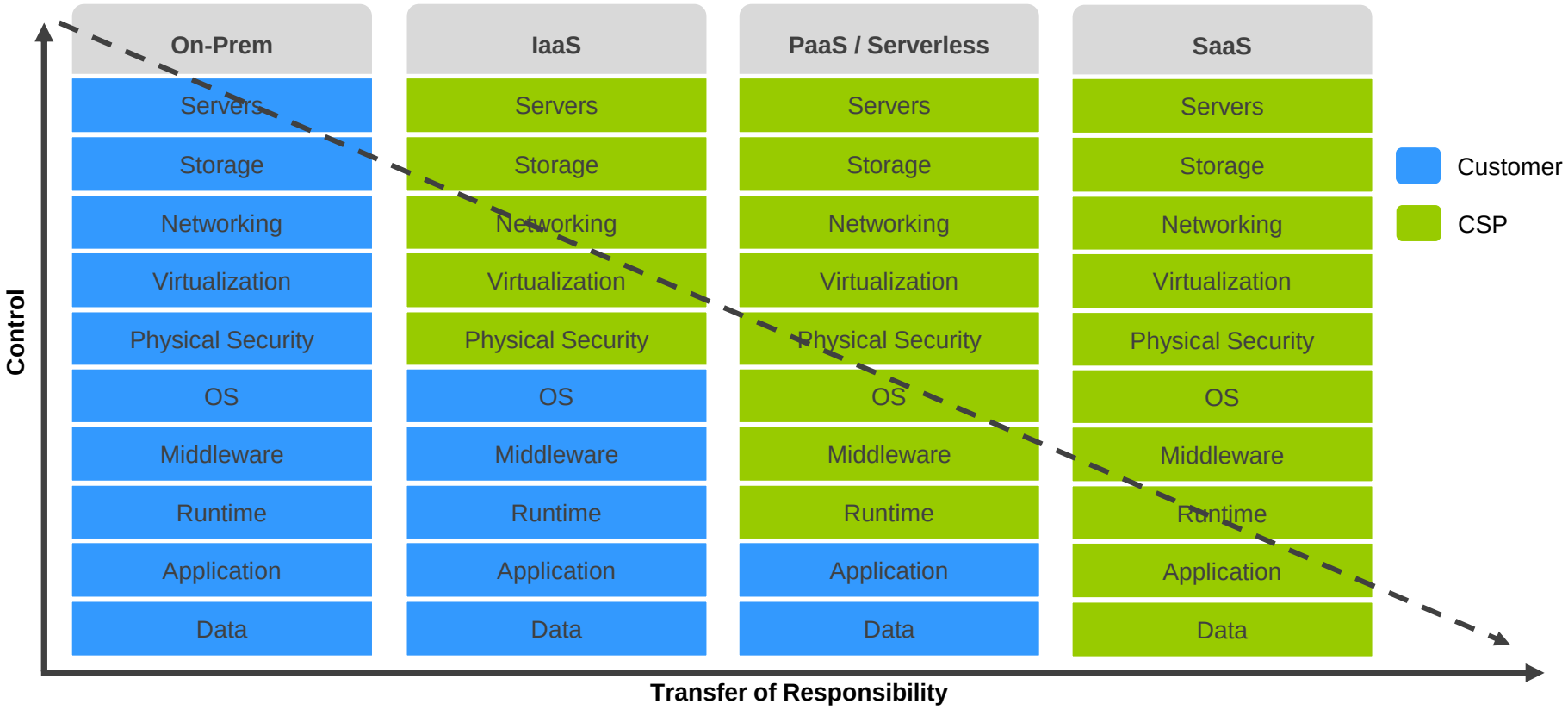
- Of those discussed, often the cheapest option for service consumers
- However, also offers minimal (or no) control, outside of exposed configuration capabilities

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

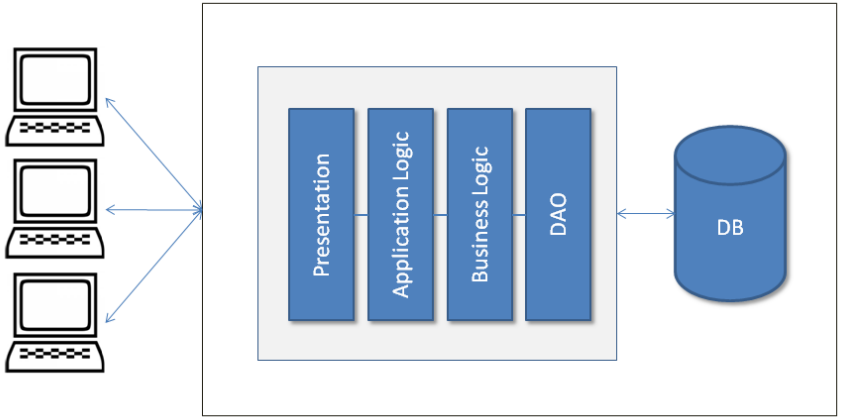


Discussion: Monolith vs. Microservices

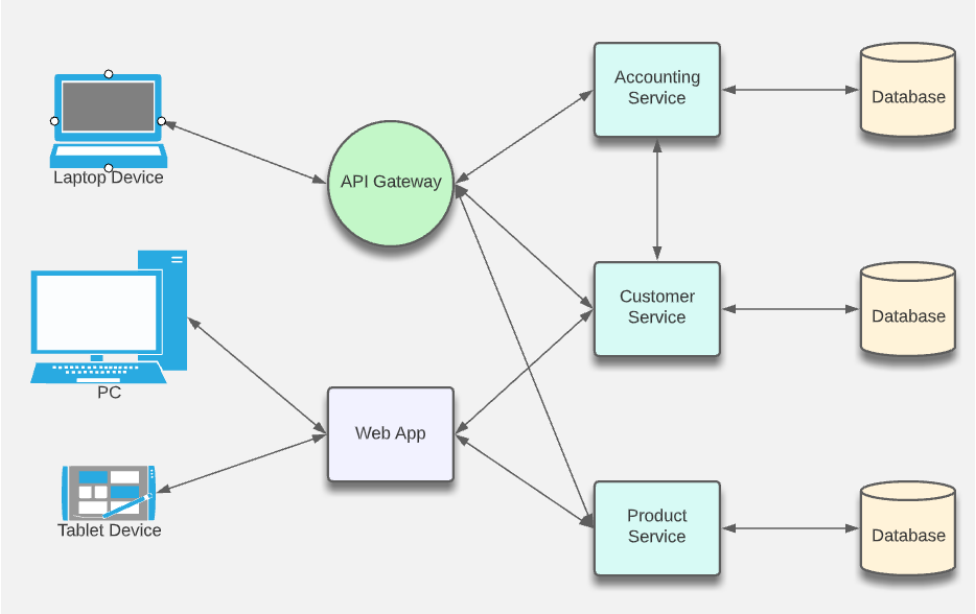
- What is a “monolith”?
- What is a “microservice”?
- What are some of the benefits of a “microservice”?
- What are some implications with a “microservice” architecture?

What is a Monolith?

- Typical enterprise application
- Large codebases
- Set technology stack
- Highly coupled elements
- Whole system affected by failures
- Scaling requires the duplication of the entire app
- Minor changes often require full rebuild



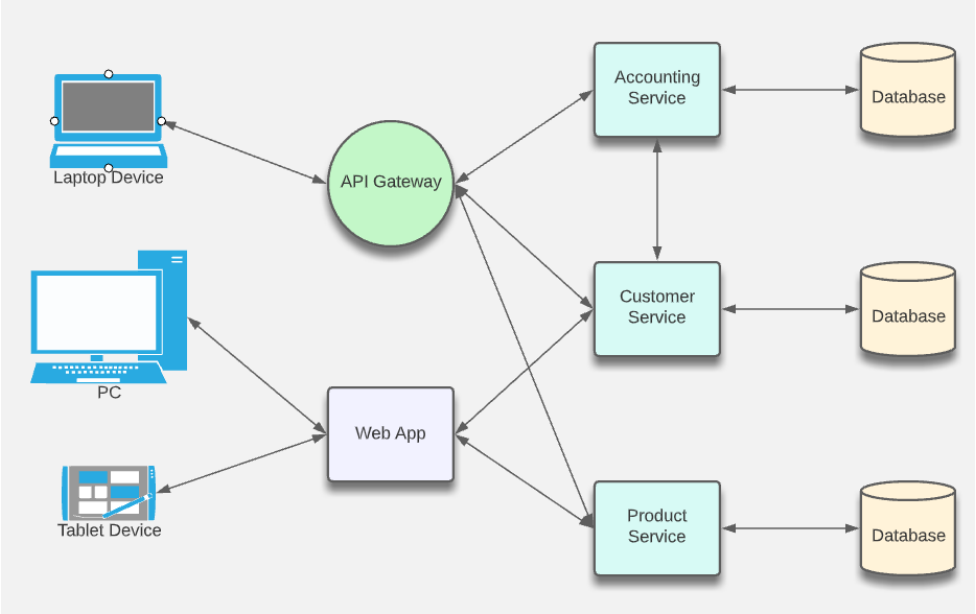
Microservices Architecture



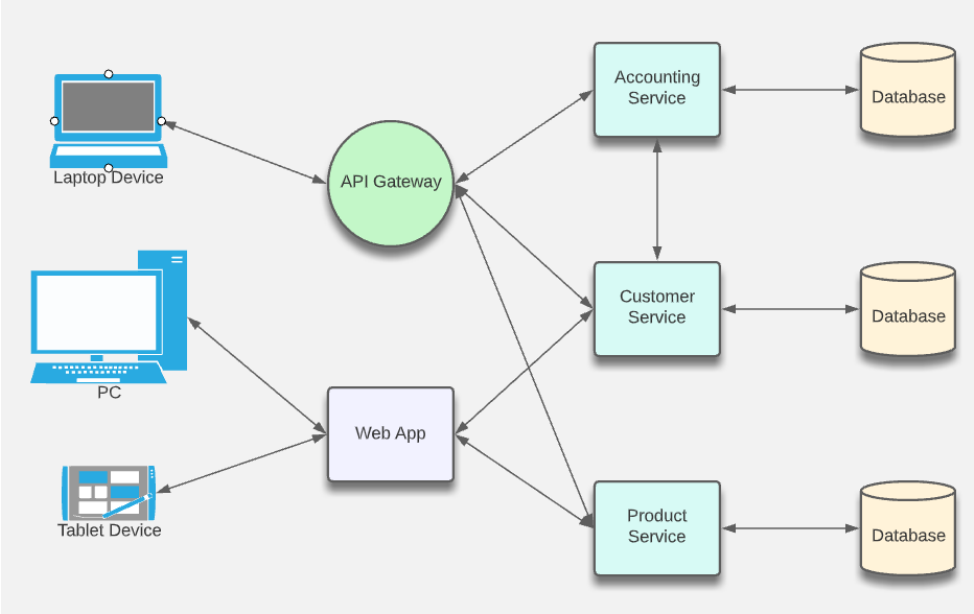
Microservices Architecture

Benefits

- Enables work in parallel
- Promotes organization according to business domain
- Advantages from isolation
- Flexible in terms of deployment and scale
- Flexible in terms of technology
- Allows multiple agile teams to maximize autonomy in effort and technology



Microservices Architecture



Implications

- Requires a different way of thinking
- Complexity moves to the integration layer
- Organization needs to be able to support re-org according to business domain (instead of technology domain)
- With an increased reliance on the network, you may encounter latency and failures at the network layer
- Transactions must be handled differently (across service boundaries)

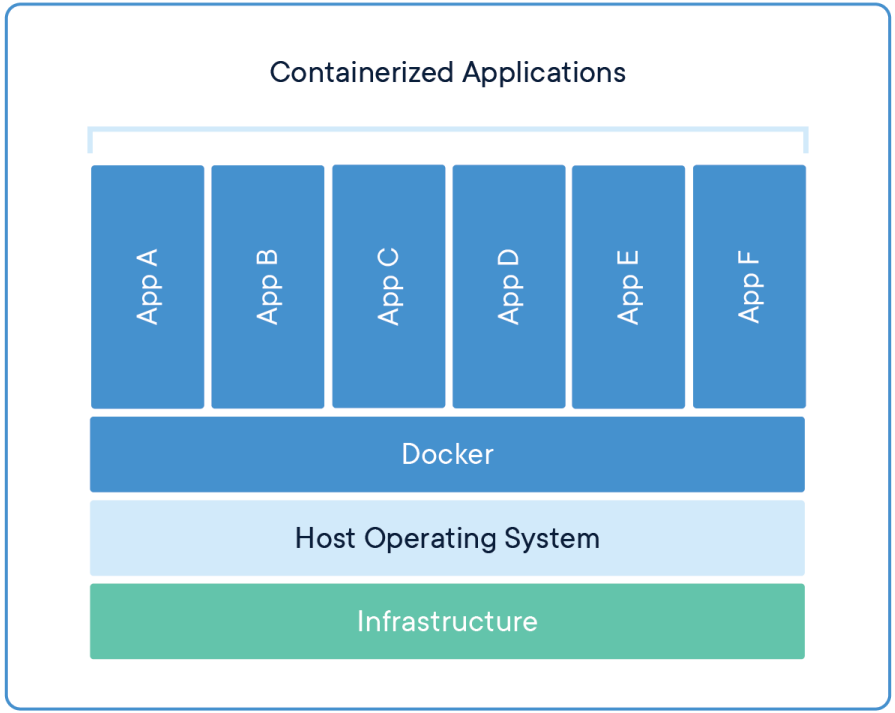
So, What Are Containers?

- Form of virtualization at the app packaging level (like virtual machines at the server level)
- Isolated from one another at the OS process layer (vs VM's which are isolated at the hardware abstraction layer)
- Images represent the packaging up of an application and its dependencies as a complete, deployable unit of execution (code, runtime and configuration)

So, What Are Containers?

- A platform (e.g., Docker) running on a system can be used to dynamically create containers (executable instances of the app) from the defined image
- Typically, much, much smaller than a VM which makes them lightweight, quickly deployable and quick to “boot up”
- An orchestration engine (e.g., Kubernetes) might be used to coordinate multiple instances of the same container (or a “pod” of containers) to enable the servicing of more concurrent requests (scalability)

So, What Are Containers?



So, Why Containers?

- Abstracts away the code implementation so you can deploy in a platform-agnostic manner, writing in the language of your choice
- Aligns strongly with the principles and practices of DevOps
- Helps leverage the power of the cloud
- Speeds up important non-coding activities (infrastructure spin-up, testing, CI/CD tasks, DevSecOps, code quality checks, etc.
- Helps breed consistency vs. “snowflake”

So, How Do Containers & Microservices Fit Together?

- Microservices – with their smaller size, independently-deployable and independently-scalable profile, and encapsulated business domain boundary – are a great fit for containers
- Using Kubernetes, sophisticated systems of integrated microservices can be built, tested and deployed
- Leveraging the scheduling and scalability benefits of Kubernetes can help an organization target scaling across a complex workflow in very granular ways
- This helps with cost management as you can toggle individual parts of the system for optimized performance

So, How Do Containers & the Cloud Fit Together?

- One option includes standing up VM's (IaaS) and installing / managing a Kubernetes cluster on those machines or
- Another option includes leveraging a managed service (PaaS) provided by the CSP
- Options in AWS include Elastic Container Service (ECS), Elastic Container Registry (ECR), and EKS (Elastic Kubernetes Service)

Stateless Applications and Scaling

- If state is held in your application, scaling is very hard. Why?
- Where can we get state info if it’s not held in the application?
- If we build to resist failure, scaling is hard. Why?
- How can we build to embrace failure?

Modern Approaches to Application Development

- Leverage the cloud
- Abstract away as much of the non-code elements as possible
- Use the tools that arise from DevOps (automation, feedback, etc.)
- Integrate testing and code quality checks early (“Shift Left”)

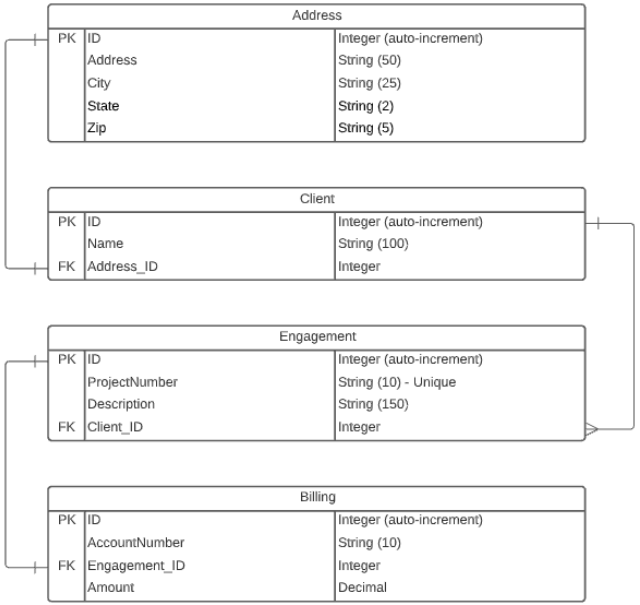
Design Approaches to Modern Applications

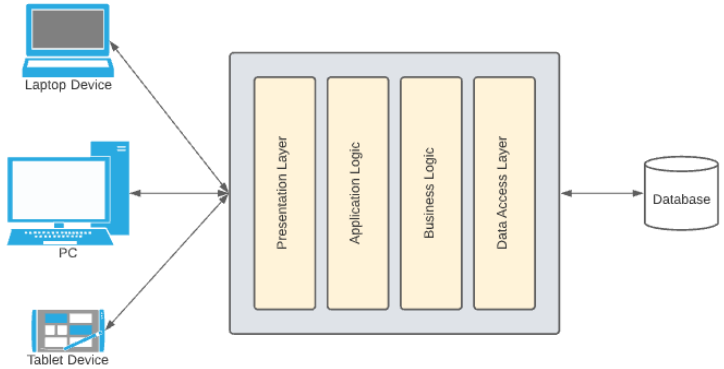
- Decompose when possible
- Embrace Failure
- Separate the language from the function of the code

Infrastructure as Code (IaC)

- Declarative vs. Imperative coding
- Drawbacks of declarative infrastructure
- Impact of IaC on the dev process, on testing, and on deployment

Use Case Discussion – Monolith to Microservices





Operations Supported:

- Add a new client (with address)
- Update an existing client (with address)
- "Soft" delete a client (with address)

- Add a new engagement associated to a client
- Update an existing engagement associated to a client
- "Soft" delete an engagement - client remains unchanged

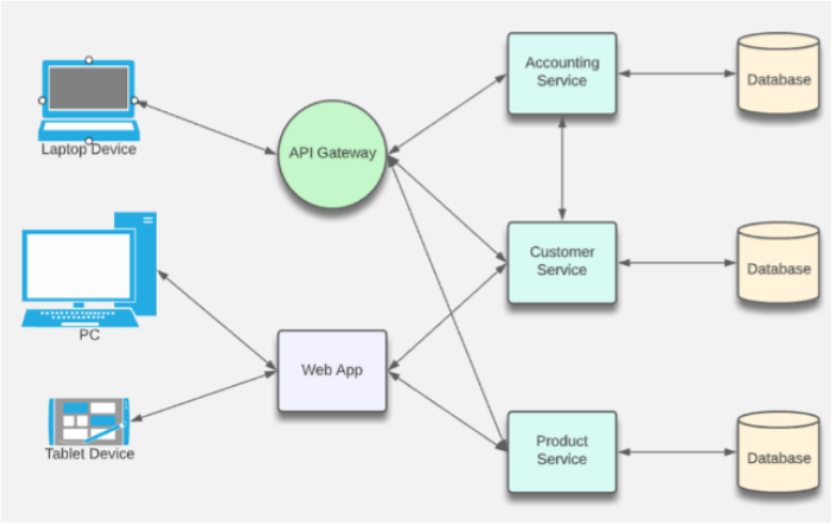
- Add a new billing entry associated to an engagement
- Updates and deletes are not allowed - to adjust, apply compensating transactions as required

- Ability to view all entities - list of customers (with address), list of engagements for a client, and list of billing entries for an engagement
- Using the system, users can initiate the generation of a batch report - this operation will generate and send a file to the presentation layer for formatted display

Use Case Discussion – Exercise

Using a collaborative visualization tool of your choice, as a team in your breakout room, discuss and formulate an approach to transitioning the client engagement monolith to a microservices architecture. You can adjust the organization of your tables for data storage as you see fit and in alignment with your microservices strategy. The goal is to separate features and functions within the workflow to help optimize advantages provided by a microservices architecture - correctly aligned to business domain, ability to work autonomously and in parallel as teams, good architecture and loose coupling, independent deployability, and independent scalability.

Clearly represent your databases for each microservice on the diagram (e.g. using). Use a separate shape to represent your microservices and use text to describe which of the supported functions will be taken on by a given service.



Demo: Microservices & Containers

Reference Application: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/microservice-application-design

Repo: https://github.com/dotnet-architecture/eShopOnContainers

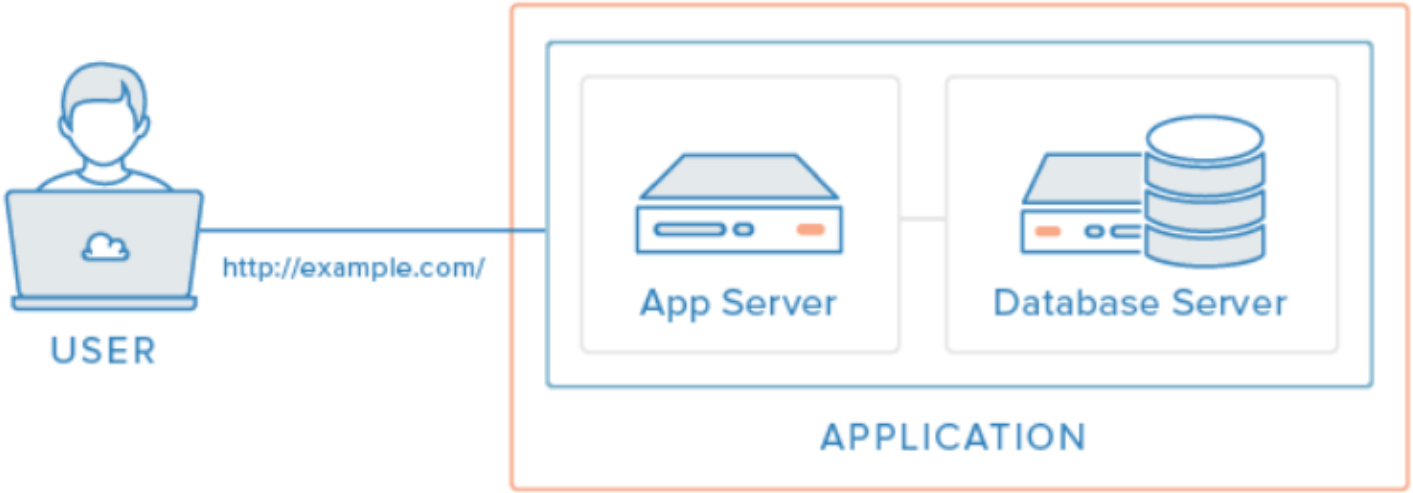
Set up / Verify Local k8s Dev Environment

- Activate Kubernetes now, via Docker Desktop
- Use kubectl from command line to see k8s config data and confirm setup
- Optional: Install Docker/k8s extensions/plugins for your IDE

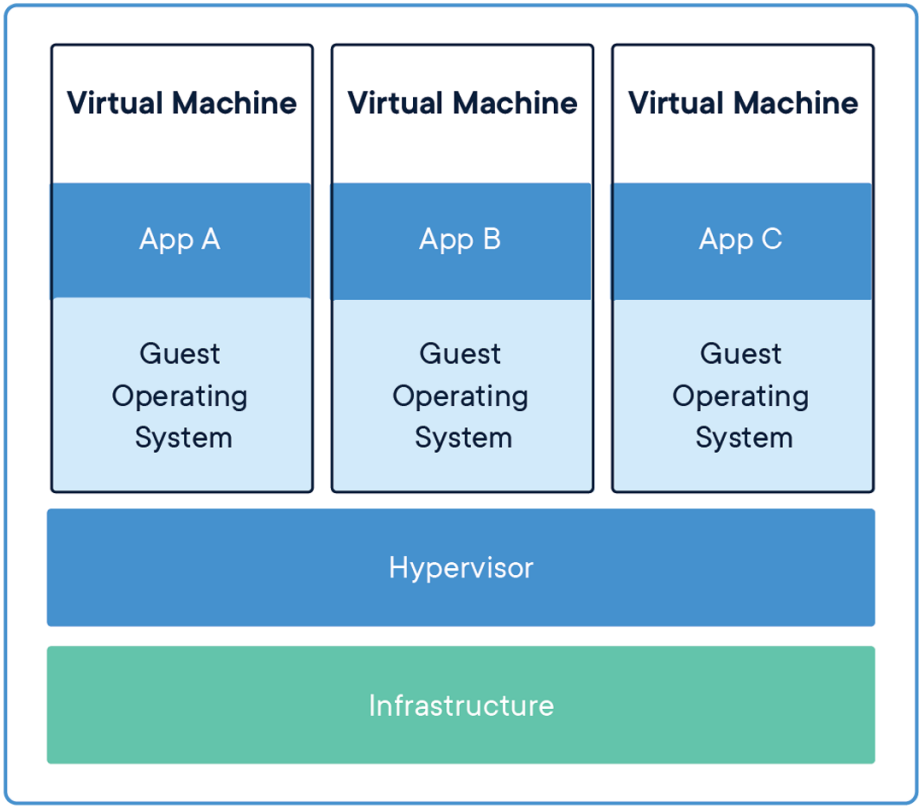
Lab 01

Introduction to Containerization & Docker

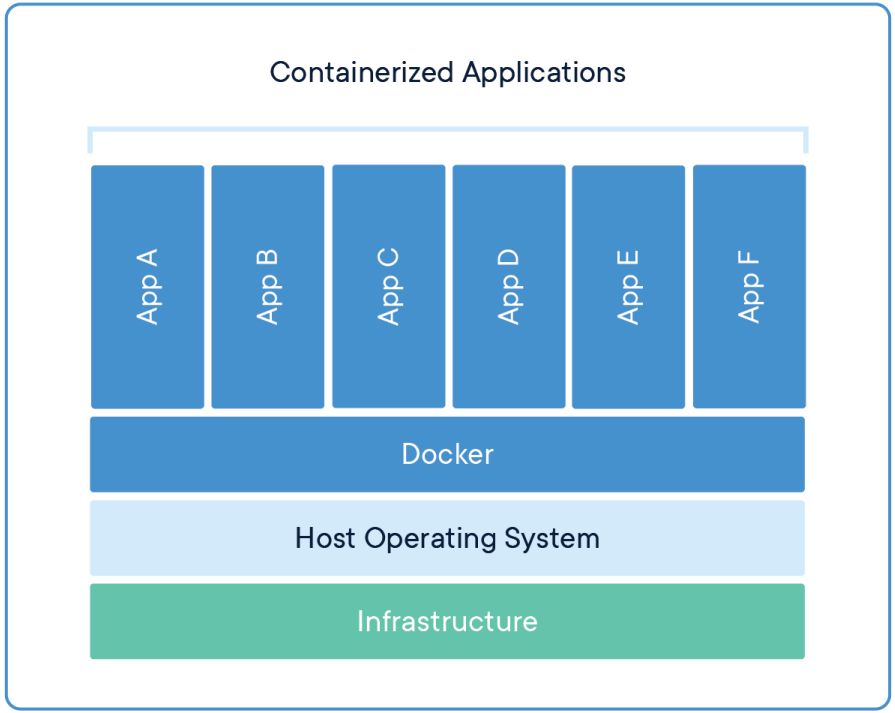
Evolution of Containers – Client/Server



Evolution of Containers – Virtual Machines



Evolution of Containers – Containers



The Container Ecosystem

- Docker: The company vs. the application
- Docker client/server architecture
  - Docker daemon (dockerd)
  - Docker Client (docker)
  - Host
  - Registries (public/private)
- Docker Objects
  - Images
  - Containers
  - Networks
  - Volumes
  - Plugins

What is Docker?

- Open-source containerization technology
- Enables deployment of self-contained & isolated application instances
- One of the foundational technologies for supporting microservices

What is Docker?

- Built around the concept of images & containers
- Also, supports composition of a set of containers to be deployed together
- For example, application code + network components + database components

What is Docker?

- Utilizes principles of “immutable infrastructure”
- Complete application environments torn down and recreated as needed
- Helps to minimize infrastructure “drift” and environment inconsistencies

The Dockerfile

- Tells Docker what to do in creating an image for your application
- The commands are all things you could do from the CLI
- Used by the docker “build” command
- Docker build uses this file and a “context” – a set of files at a specified location – to make your image

Dockerfile example

The following creates an image for building/running Java app in container
See https://github.com/KernelGamut32/dockerlab-repo-sample for sample

Dockerfile

```
# Grabs OpenJDK image upon which the new image will be based
FROM openjdk:17

# Creates a new target folder in image
RUN mkdir /usr/src/JavaDemoApp

# Copies current directory contents to newly created folder
COPY . /usr/src/JavaDemoApp

# Switches working directory in image to app folder
WORKDIR /usr/src/JavaDemoApp

# Compiles/builds Java app
RUN javac JavaDemo.java

# Executes new Java app
CMD ["java", "JavaDemo"]
```

Docker Images

- Represent templates defining an application environment
- New instances of the application can be created from the image
- These instances are called containers

Docker Images

- Images are defined via a Dockerfile definition
- Support layers for building up the environment in stages
- Fully defines the application, including all components required to support

Docker Images

Those components can include:

- Runtime
- Development framework
- Source code
- Executable instructions for container startup

Docker Images

- Start with a base that gives your app a place to live
  - Needed OS/runtimes/dB server applications, etc.
- Examples:
  - nginx
  - Node
  - MySQL
  - Apache HTTP Server
  - IIS with .NET Runtimes

Docker Hub

- Centralized registry for image storage & sharing
- Can signup for an account – user accounts offer both free and pro versions
- Also, supports organizations for grouping of multiple team members

Docker Hub

- Accessible at https://hub.docker.com
- Search feature enables search for image by technology or keyword
- Image detail displays available tags and image variants

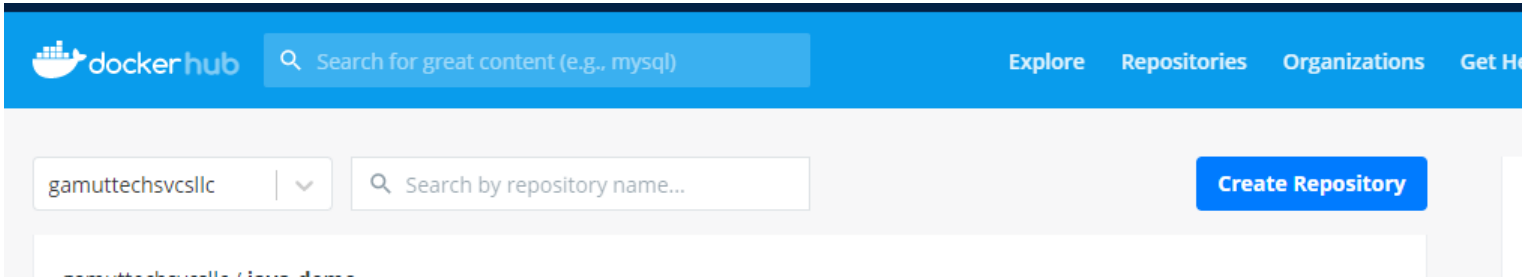
Docker Hub

- May also include examples of usage
- Pro account supports image scanning for security vulnerabilities
- Can be useful for image reuse and image sharing across a dev team

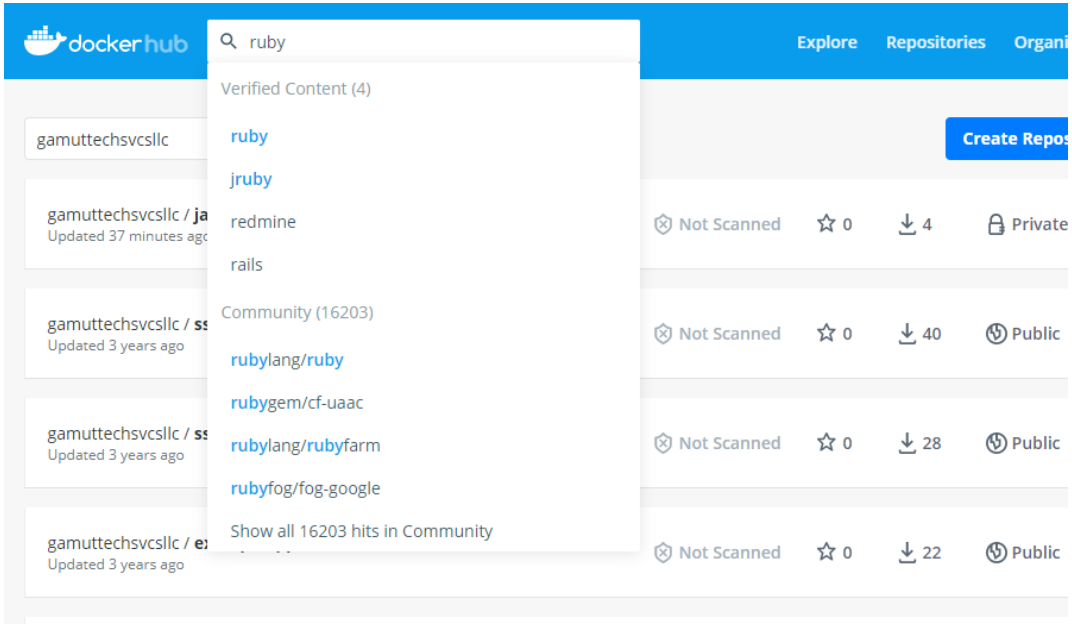
Docker Hub

There are other registry types, including private registries

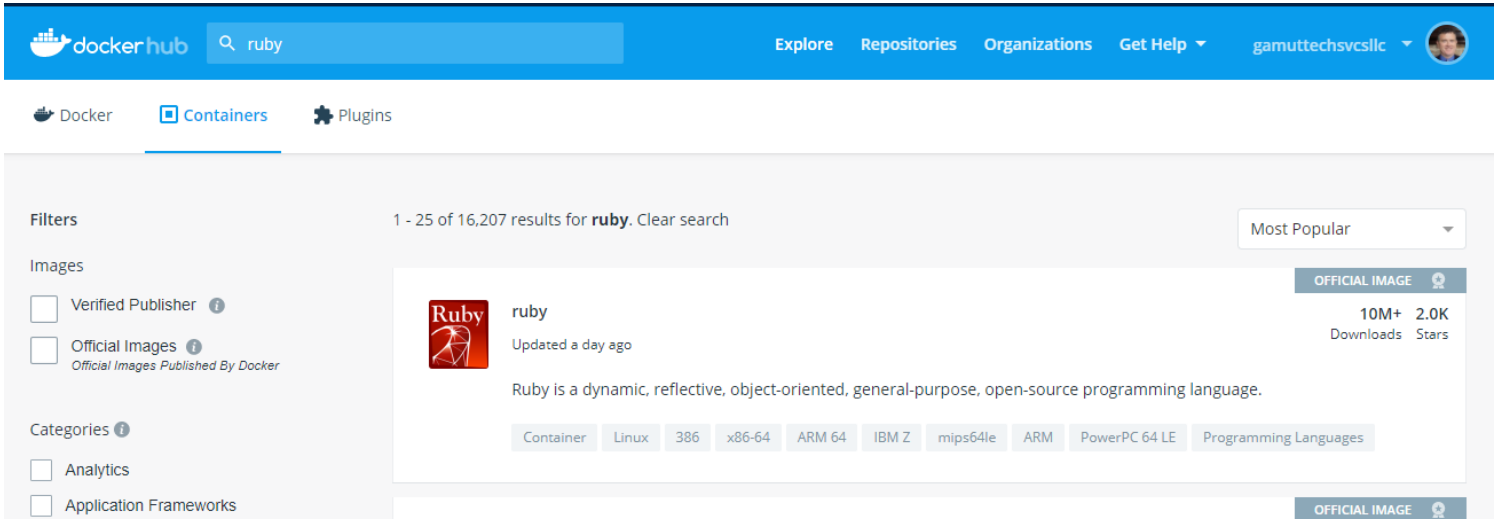
Docker Hub


dockerhub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Get H
gamuttechsvcsllc
Search by repository name...
Create Repository

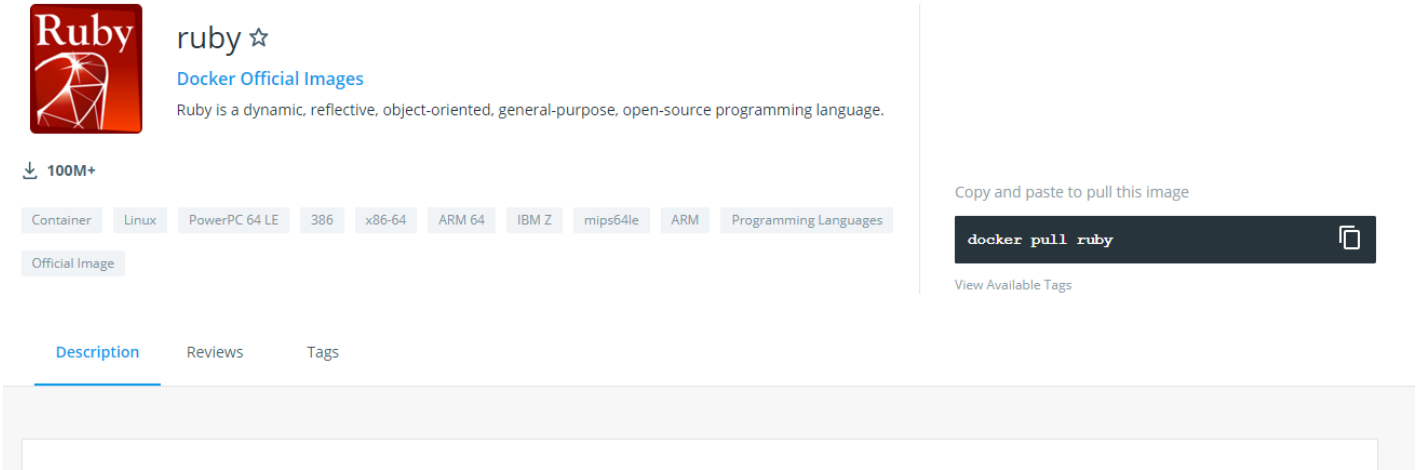
Docker Hub


docker hub
ruby
Explore
Repositories
Organi
Verified Content (4)
ruby
jruby
redmine
rails
Community (16203)
rubylang/ruby
rubygem/cf-uaac
rubylang/rubyfarm
rubyfog/fog-google
Show all 16203 hits in Community
gamuttechsvcsllc
Create Repos
gamuttechsvcsllc / ja
Updated 37 minutes ago
Not Scanned
0
4
Private
gamuttechsvcsllc / ss
Updated 3 years ago
Not Scanned
0
40
Public
gamuttechsvcsllc / ss
Updated 3 years ago
Not Scanned
0
28
Public
gamuttechsvcsllc / ex
Updated 3 years ago
Not Scanned
0
22
Public

Docker Hub


docker hub
ruby
Explore
Repositories
Organizations
Get Help
gamuttechsvcsllc
Docker
Containers
Plugins
Filters
Images
Verified Publisher
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
1 - 25 of 16,207 results for ruby. Clear search
Most Popular
OFFICIAL IMAGE
ruby
Updated a day ago
10M+ Downloads
2.0K Stars
Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.
Container
Linux
386
x86-64
ARM 64
IBM Z
mips64le
ARM
PowerPC 64 LE
Programming Languages
OFFICIAL IMAGE

Docker Hub


Ruby
ruby ☆
Docker Official Images
Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.
100M+
Container
Linux
PowerPC 64 LE
386
x86-64
ARM 64
IBM Z
mips64le
ARM
Programming Languages
Official Image
Copy and paste to pull this image
docker pull ruby
View Available Tags
Description
Reviews
Tags

Docker Hub

How to use this image

Create a `Dockerfile` in your Ruby app project

```
FROM ruby:2.5

# throw errors if Gemfile has been modified since Gemfile.lock
RUN bundle config --global frozen 1

WORKDIR /usr/src/app

COPY Gemfile Gemfile.lock ./
RUN bundle install

COPY . .

CMD ["./your-daemon-or-script.rb"]
```

Put this file in the root of your app, next to the `Gemfile`.

You can then build and run the Ruby image:

```
$ docker build -t my-ruby-app .
$ docker run -it --name my-running-script my-ruby-app
```

Generate a `Gemfile.lock`

Building The Image

- To build the image from Dockerfile use *docker build*
- *docker build -t <tag name> <path to Dockerfile>*
- For example, *docker build -t java-demo .*
- Builds image from Dockerfile in current folder (.) with tag name “java-demo”

Building The Image

- For tag name, can include optional detail:
  - Docker ID in Docker Hub for eventual push to image registry
  - Version identifier for tag – defaults to “latest” if excluded
- For example, *docker build -t <docker ID>/<tag name>:<version> .*

Pushing/Pulling Image

- Use *docker push <docker ID>/<tag name>:<version>* to push to registry
- Use *docker pull <docker ID>/<tag name>:<version>* to pull from registry
- May prompt for credentials (e.g., Docker Hub login)

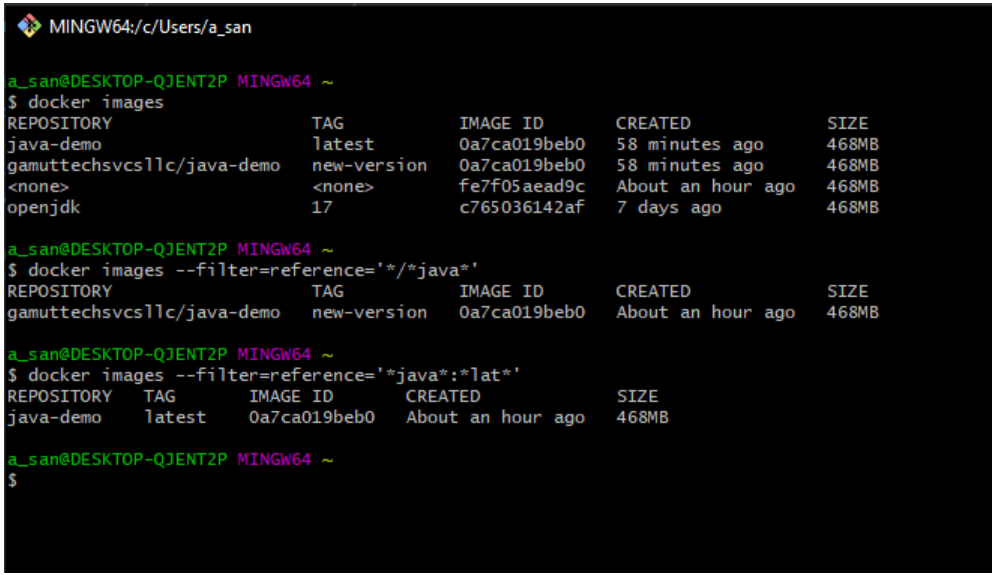
Pushing/Pulling Image

- To pull from public registry, use docker pull <tag name>:<version>
- For example, *docker pull openjdk:17*
- A .dockerignore file can be used to omit files/folders on push

Managing Images

- Use *docker images* to list available local images
- *--filter* argument enables wildcard search
- *docker images --filter=reference='<wildcard for tag name>:<wildcard for version>'*

Managing Images



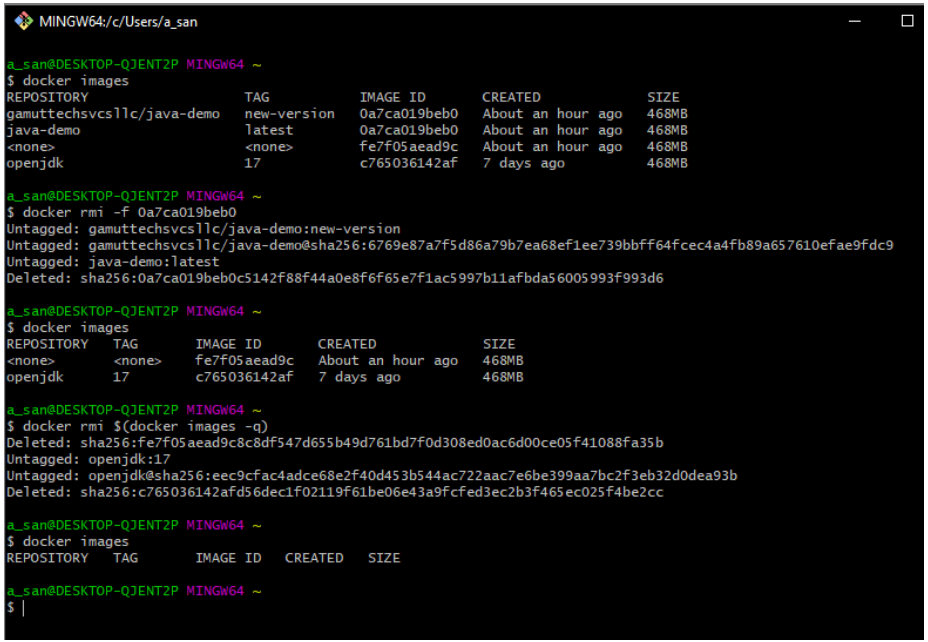
Managing Images

- To remove an image, use *docker rmi*
- *-f* argument used to remove images even when used by containers
- Can use *docker rmi -f <image ID>* to remove specific image

Managing Images

- Can use *docker rmi -f $(docker images -q)* to remove all (CAUTION)
- *docker images -q* lists images in quiet mode (returns image ID’s only)

Managing Images



Layers in a Docker Image

- Each instruction in a dockerfile makes a “layer”
- It’s actually a diff from the earlier layer
- Allows Docker to skip redundant info and use cached artifacts
- In this way, images themselves are actually diffs – they show what changed from the earlier stage (e.g., our app’s image is actually a diff from the base image you chose)

Best Practices for Creating Docker Images

- Single app per container
- Don’t include unnecessary tools in your image (dev tools; network tools like netcat, etc.)
- Build as small an image as possible
  - Choose a small base image
  - Optimize your app for image size
- Use a consistent “tag” strategy
  - Document it
  - Use it for version info, testing strategy info, etc.
- Be smart about using public base images

Troubleshooting Docker Images

- Most common area is the Dockerfile
- Error on docker build has good messaging and an error code

Docker Containers

- Represent “runnable” instances of a docker image
- Application instance created from image in container can be used as:
  - Isolated executable
  - Request servicer (e.g., web listener)

Docker Containers

- Includes all dependencies and runtime defined by the image
- Isolated from other containers at the OS process layer
- Mechanisms exist to share resources across containers (e.g., files or DBs)
- However, isolation is what makes them powerful – avoid unnecessary coupling

Docker Containers

- Typically, containers are much smaller which makes them lightweight
- Quickly deployable and quick to “boot up”
- Isolation allows technologies like k8s to spin up multiple as needed
- “Load balancers” route to any of multiple instances using single point of connection

Creating Containers

- To create a new container, you can use *docker create*
- Multiple options are provided for configuring container (see *docker create --help*)
- For example, *docker create --name <container name> <image tag>*
- *<image tag>* defines image (template) from which to create container

Creating Containers

- To list available containers, use *docker ps*
- *docker ps -a* lists all containers (even those not currently started)
- Containers can be stopped and started

Creating Containers

- Use *docker start <name>* or *docker start <container ID>* to start
- Use *docker stop <name>* or *docker stop <container ID>* to stop
- Use *docker run* to create and start container in single step
- *docker run* is the more common command

Configuring Containers

- Command-line options for configuring containers using *docker run* include:
  - --*name* <*container name*> – give container user-defined name
  - -*p* <*host port*>:<*container port*> – map host port to container port for access
  - -*it* – indicates interactive on command-line (e.g., for gathering command-line input)
  - --*rm* – container automatically deleted when it exits or stops
  - -*d* – container runs in detached mode (e.g., for continually running web listeners)
- See *docker run --help* for additional info

Container Status

- Use *docker logs <container name>* to see log output from container
- Use *docker logs -f <container name>* for ongoing monitor of log output
- Helpful for troubleshooting issues with container creation, startup, or operation

Container Commands

- *docker exec* can be used to execute a command in a running container
- For example, *docker exec <container name> ls -a* to see container file contents
- *docker exec -it <container name> /bin/bash* for interactive command-line session in container (for Linux-based images that support bash)

Managing Containers

- *docker rm <container name>* or *docker rm <container ID>* removes container
- *-f* argument used to remove a running container
- Can use *docker rm -f $(docker ps -aq)* to remove all (CAUTION)
- *docker ps -aq* lists all containers in quiet mode (returns container ID's only)

Data Storage in Docker

- Running containers generate data
- They may need access to “persistent” data
- We can use the host machine via a “bind mount” – mount a local file or folder into a container
  - Problems: Not easily managed by docker CLI
  - Rely on the host machine having a specific directory structure
- Better is to use a docker construct called a “volume”
  - New directory created in the host machine’s docker storage directory
  - Easier to back up
  - Work the same on both Linux and Windows containers
  - Can be safely shared between containers
  - Content can be pre-populated by the container

Common Docker Issues – How to Identify and Fix

- Dependency issues with base image:
  - `RUN apt-get clean && apt-get update (clears cache in event base image has been updated in the registry)`
- You may need to do this outside the container as well
- Container naming collisions:
  - If you try to use a container name that exists, you'll throw an error – EVEN if the container isn't being used. Remove it to use the name again.

Application Bootstrapping with Docker and k8s

- Kubernetes provides a hosting environment for containerized applications
- Once you have a Docker image, you can work entirely within Kubernetes to deploy your app

Open Container Initiative (OCI)

- The OCI is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes
- Docker started it
- They have two specs: runtime-spec and image-spec
- This deals with containers in the abstract

Competing Container Runtimes

- rkt from CoreOS
- Mesos from Apache
- LXC Linux containers

Lab 02

Lab 03

Kubernetes and Container Orchestration

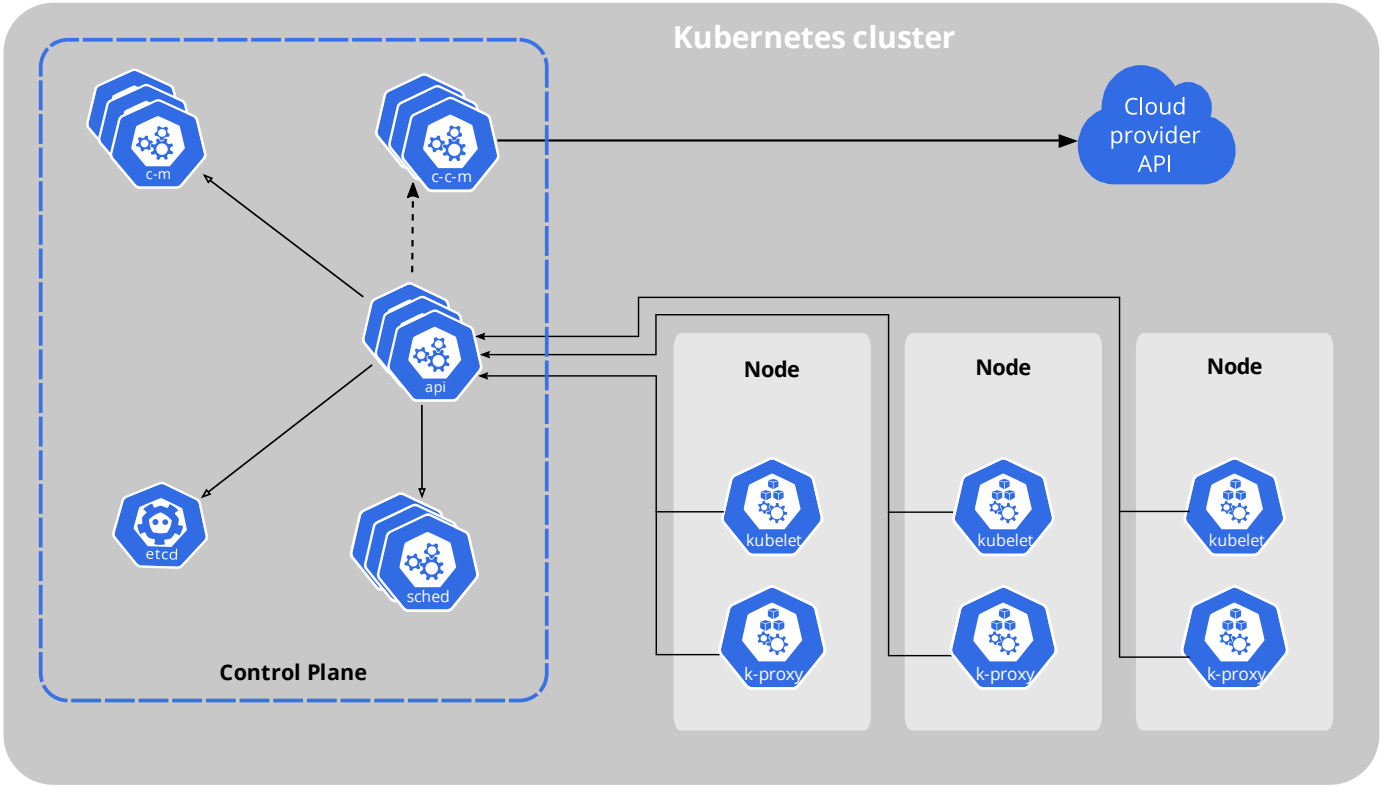
Kubernetes (k8s) Overview

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services

What is an Orchestrator and Why Do We Need It?

- What would it look like to manually control the containers needed for your application as your app scales or containers fail?
- “Orchestration” is the execution of a defined workflow
- We can use an orchestrator to start and stop containers automatically based on your set rules
- What does this open up for you?

Architecture of k8s System



API server

Cloud controller manager *(optional)*

Controller manager

etcd (persistence store)

kubelet

kube-proxy

Scheduler

Control plane

Node

Control Plane: The API

- The fundamental fabric of Kubernetes
- RESTful API
- Enables all communication between k8s components, and all operations they can do
- Enables external user commands into the cluster

Core Components of k8s

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
- The worker node(s) host the Pods that are the components of the application workload.
- The control plane manages the worker nodes and the Pods in the cluster.

k8s Resource/Manifest

- Kubernetes manifests are text files, usually written in YAML
- They are used to create, modify and delete Kubernetes resources

Kubernetes Architecture

- Kubernetes automates and monitors the lifecycle of a stateless application
- It can scale the app up or down and ensure it keeps running
- Kubernetes cluster consists of computers called nodes
- The basic unit of work in kubernetes is called a pod, which is a group of one or more containers
- Kubernetes can be divided into planes
  - Control plane - the actual pods that the kubernetes core components runs on, exposing the api, etc...
  - Data Plane - everything else meaning the nodes and pods which the application runs on

Kubernetes Architecture

- In short, control plane is what monitors the cluster, schedules the work, makes the changes, etc…
- The nodes are what actually do the real work, report back to the master, and watch for changes

Kubernetes Control Plane & Data Plane

- Kubernetes is actually a collection of pods implementing the k8s api and the cluster orchestration logic (AKA the Kubernetes control plane)
- The application/data plane is everything else, meaning all the nodes that host all the pods that the application runs on
- The *controllers* of the control plane implement control loops that repeatedly compare the desired state of the cluster to its actual state
- When the state of the cluster changes, controllers take action to bring it back inline with desired state

Declarative and Desired State

- Kubernetes supports a declarative model – we can send the api a yaml file in a declarative form
- Desired state means that we specify in the yaml file our desired state and the cluster takes responsibility to make sure it will happen
- We describe the desired state using a yaml or json file that serves as a record of intent, but we do not specify how to get there (this is kubernetes responsibility to get us there)
- Things could change or go wrong over the lifetime of the cluster (node failing, etc…), Kubernetes is responsible to always make sure that the desired state is kept intact
- Kubernetes control plane controllers are always running in a loop and checking that the actual state of the cluster matches the desired state, so that if any error occurs they kick in and rectify the cluster

Reconciling state

- Watch for the spec fields in the YAML files
- The *spec* describes *what we want the thing to be*
- Kubernetes will *reconcile* the current state with the spec (technically, this is done by a number of *controllers*)
- When we want to change a resource, we update the *spec*, and reapply
- Kubernetes will then *converge* that resource

k8s Pods: The Basic Building Block

- A pod represents a set of running containers in your cluster
- Worker nodes host pods
- The control plane manages the worker nodes and the pods inside them

k8s Pods: The Basic Building Block

- Pod is the basic execution and scaling unit in Kubernetes
- Kubernetes runs containers but always inside pods
- It is like a sandbox in which containers run (abstraction)

A pod is a group of containers:

- Running together (on the same node)
- Sharing resources (RAM, CPU; but, also, network, volumes, etc…)

- A Pod models an application-specific “logical host”

Pod state

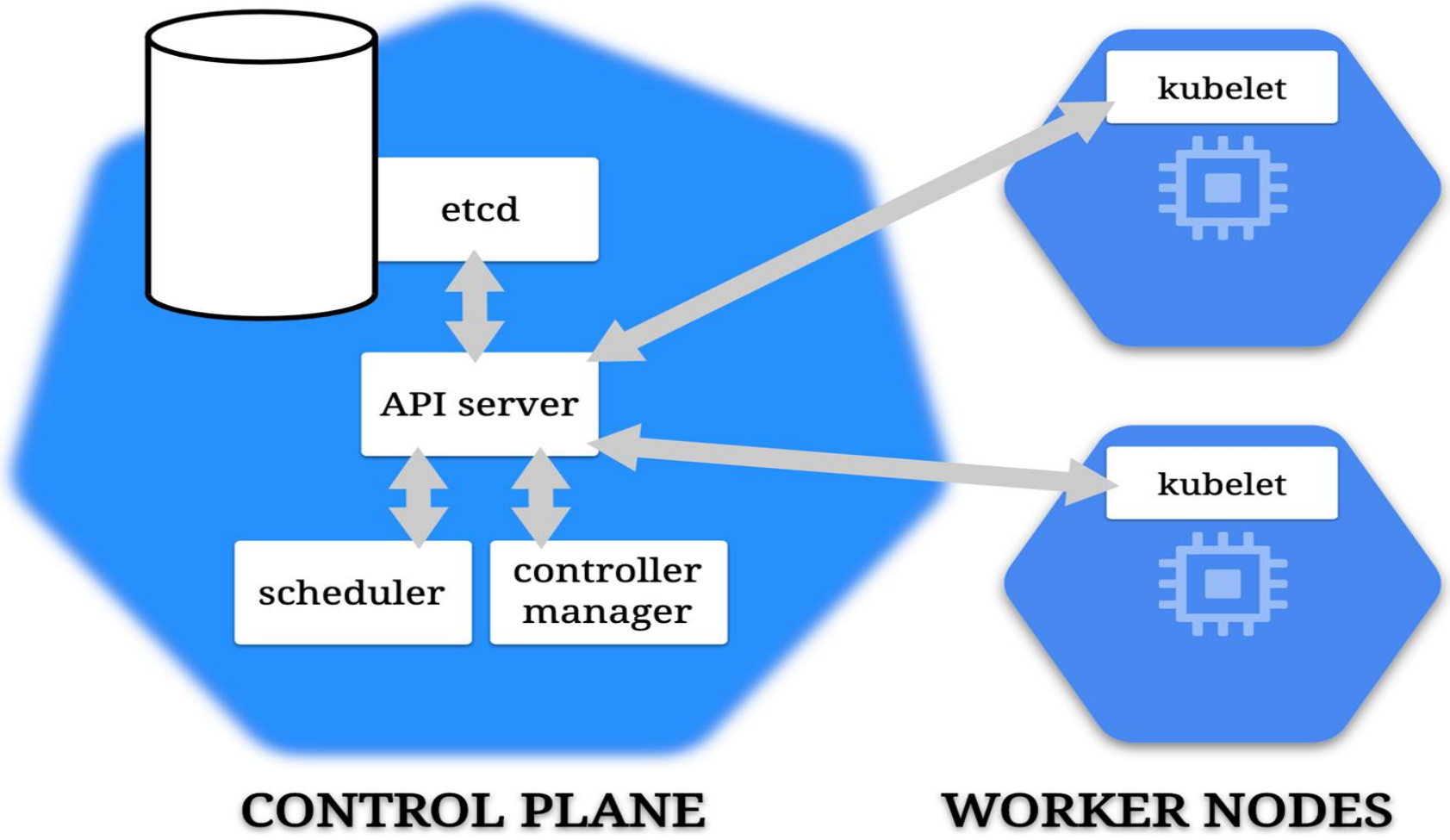
- Pods are considered to be relatively ephemeral (rather than durable) entities
- Pods do not hold state, so if a pod crashes, Kubernetes will replace it with another
- Multiple instances of the same pod are called replicas

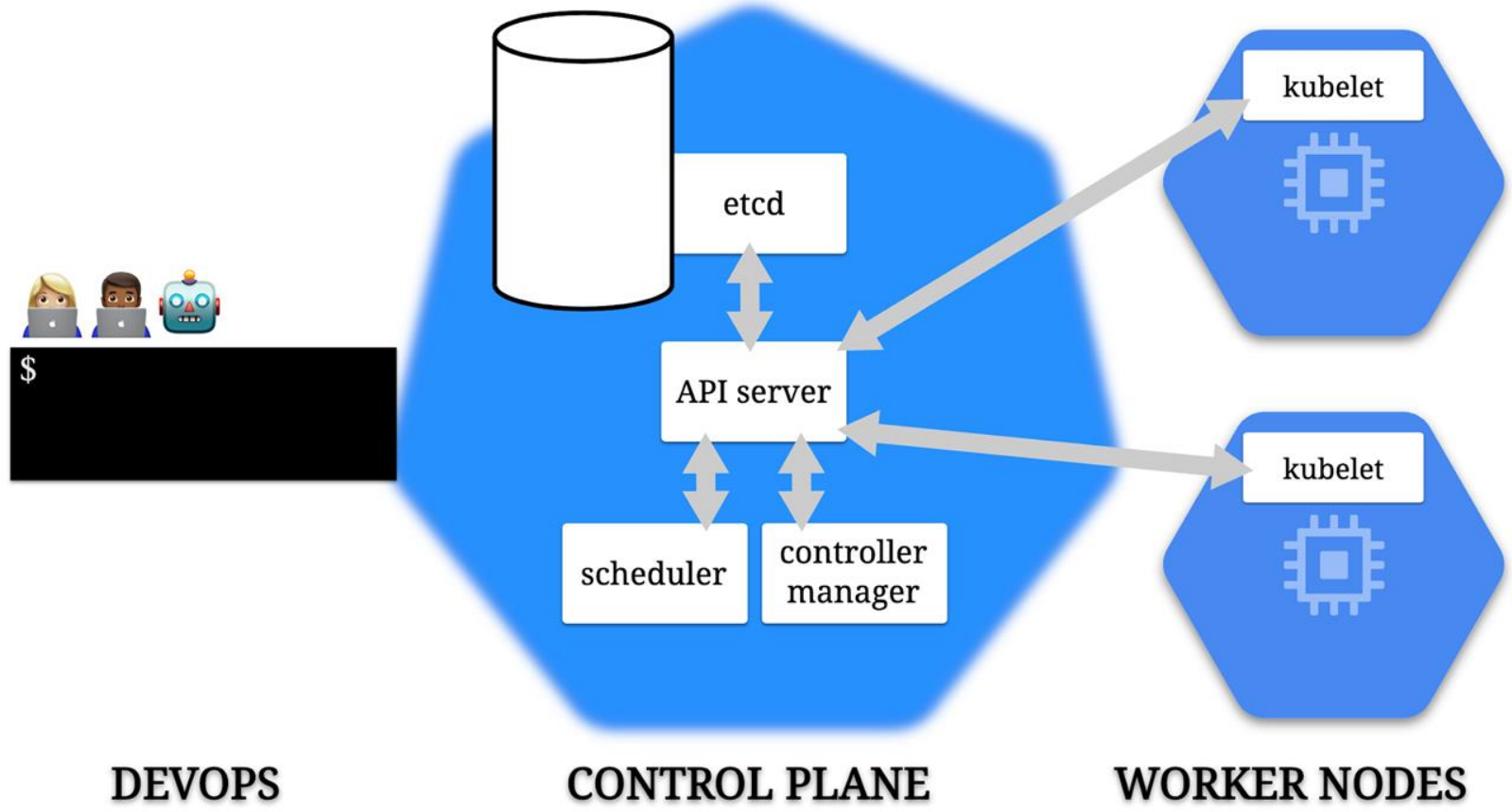
Deployments

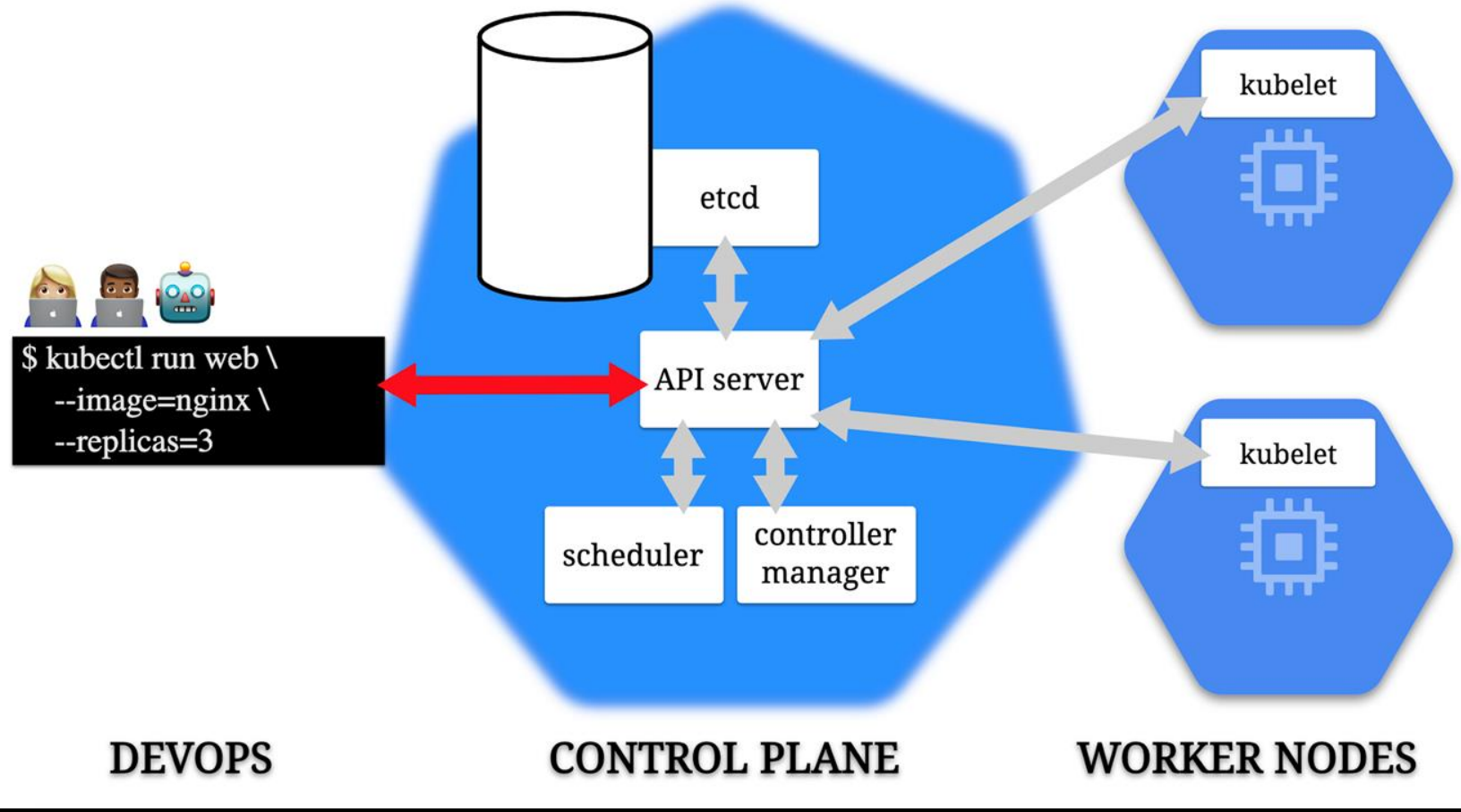
- A *Deployment* controller provides declarative updates for Pods and ReplicaSets
- The Deployment Object gives us a better way of handling the scaling of pods
- The advantage of using Deployment versus using a replicaset is having rolling updates support for the pod container versions out-of-the-box

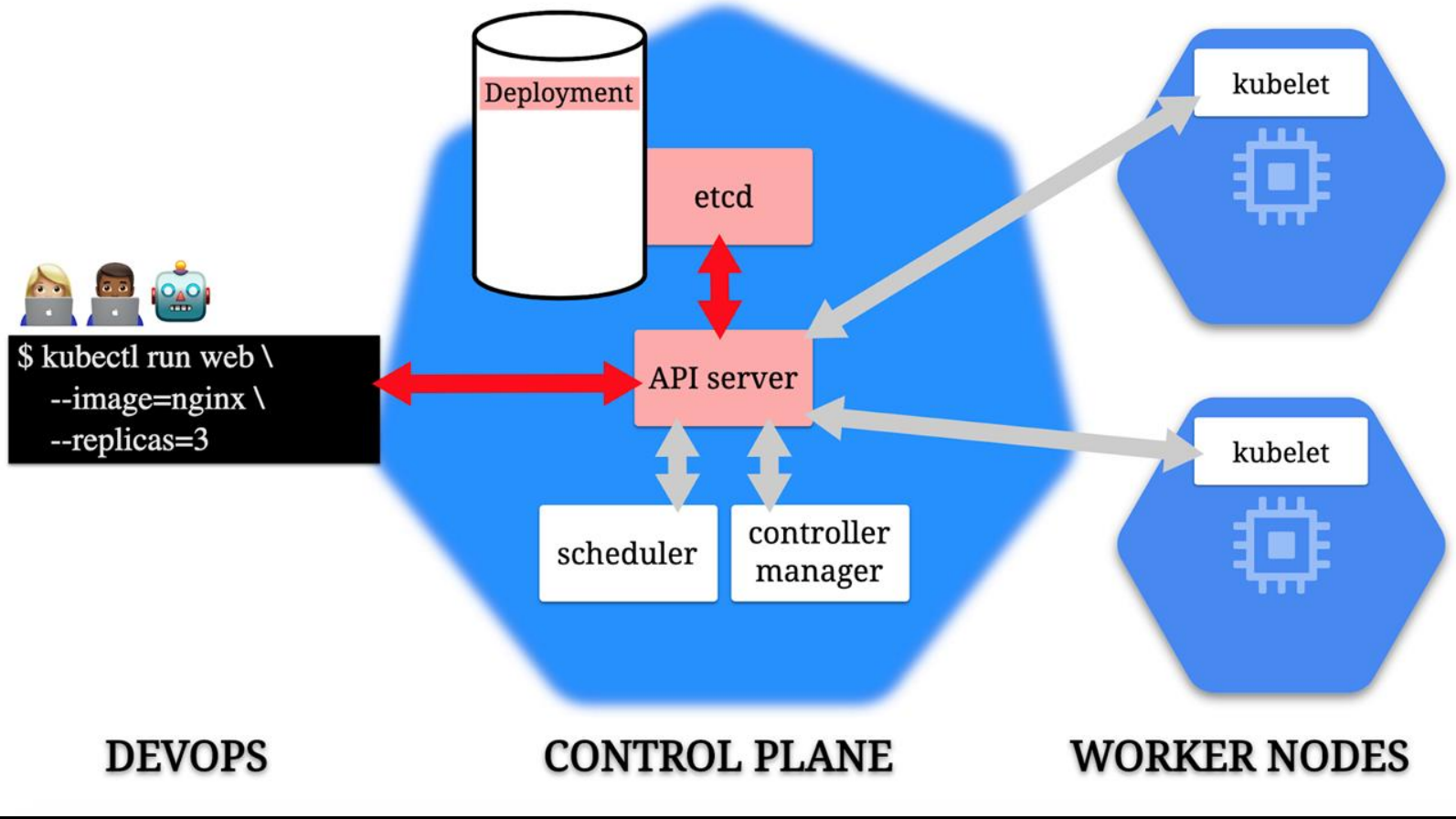
Deployments

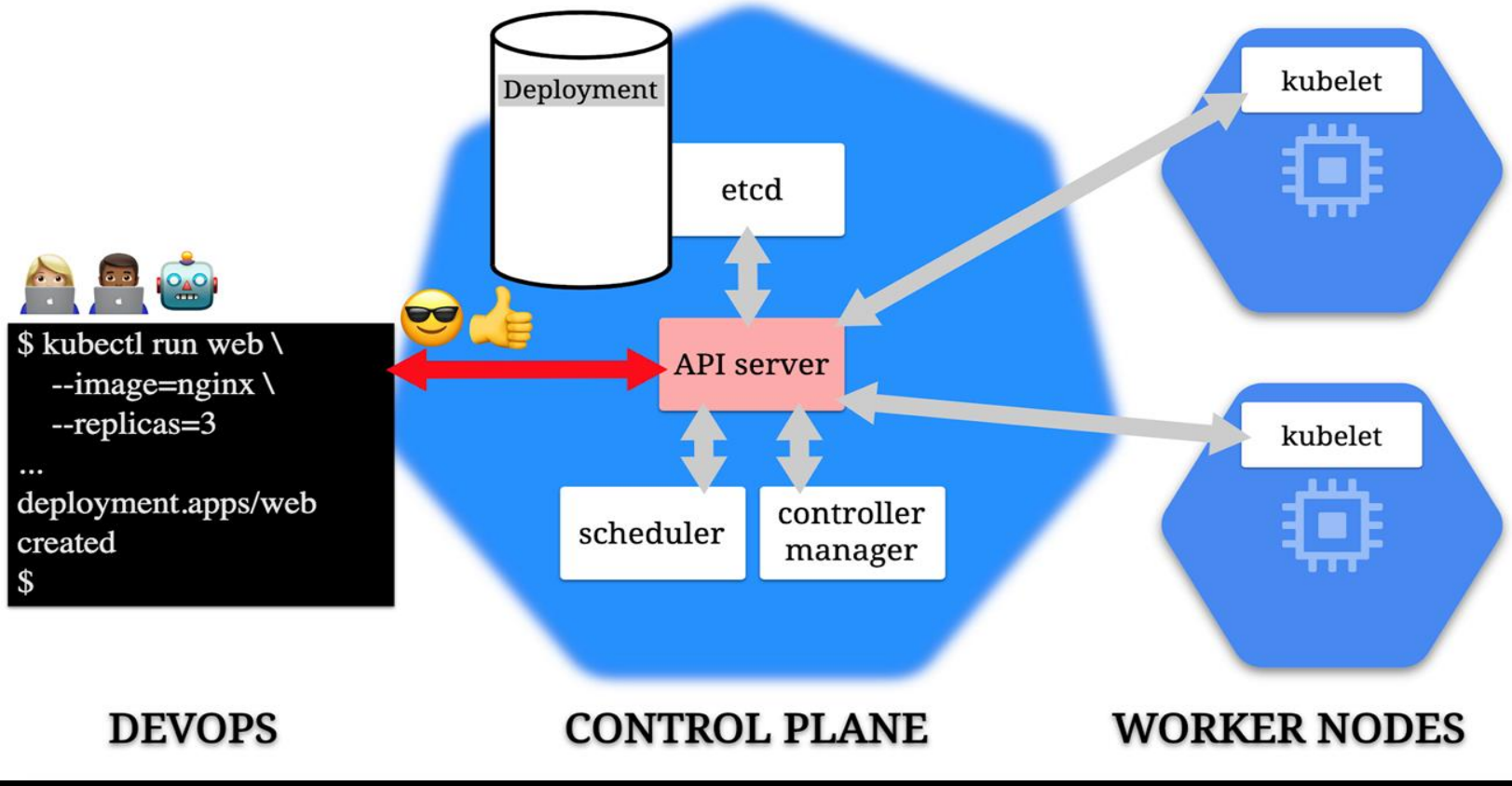
- Every time the application code changes, a new version of the application container is built, and then there is a need to update the Deployment manifest with the new version and tell K8s to apply the changes
- K8s will then handle the rolling-out of this newer version, terminating pods with the old version as it spins up the new pods with the updated container
- This means that at some point we will have multiple versions of the same application running at the same time

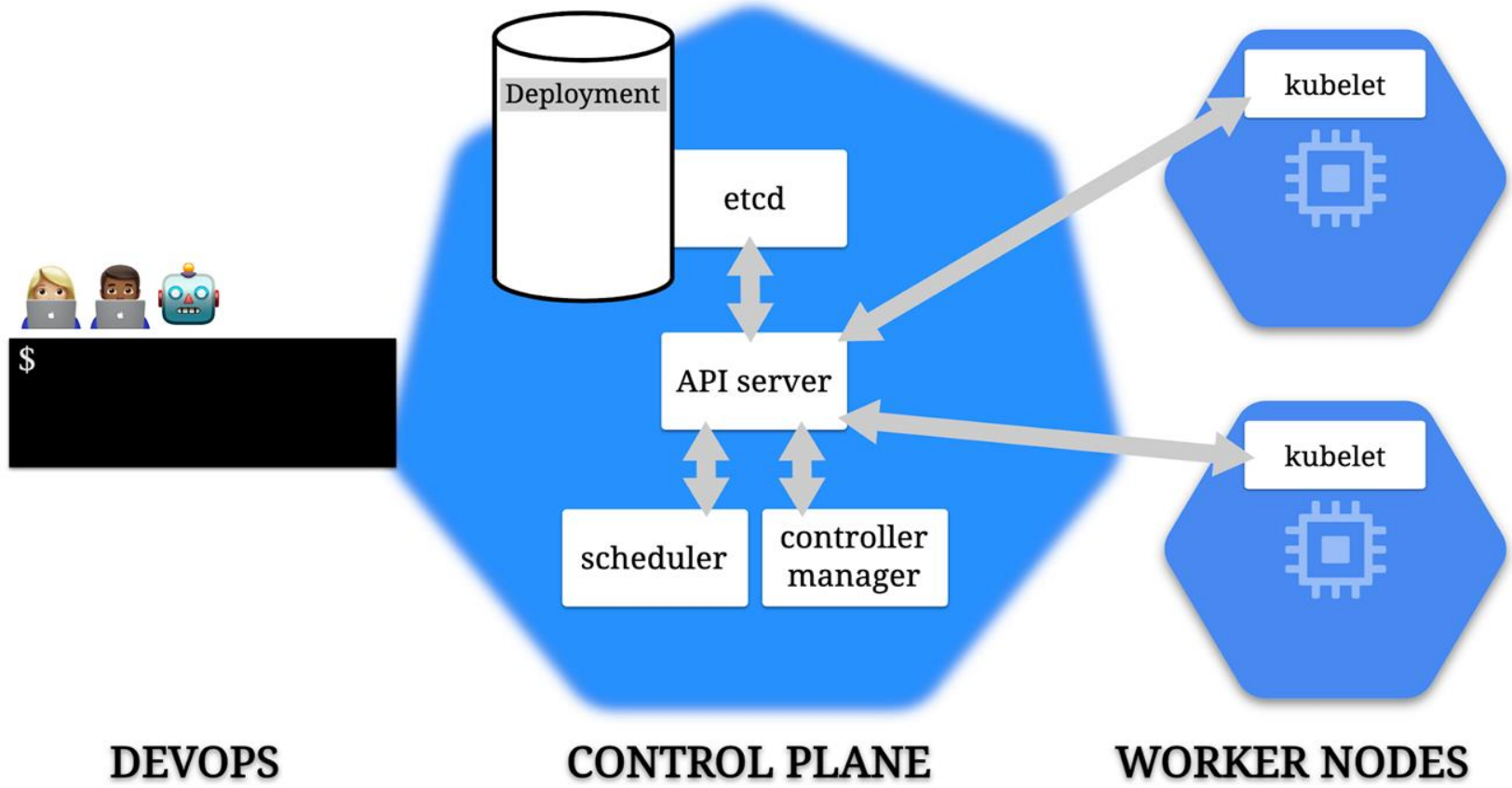

etcd
API server
scheduler
controller manager
kubelet
kubelet
CONTROL PLANE
WORKER NODES

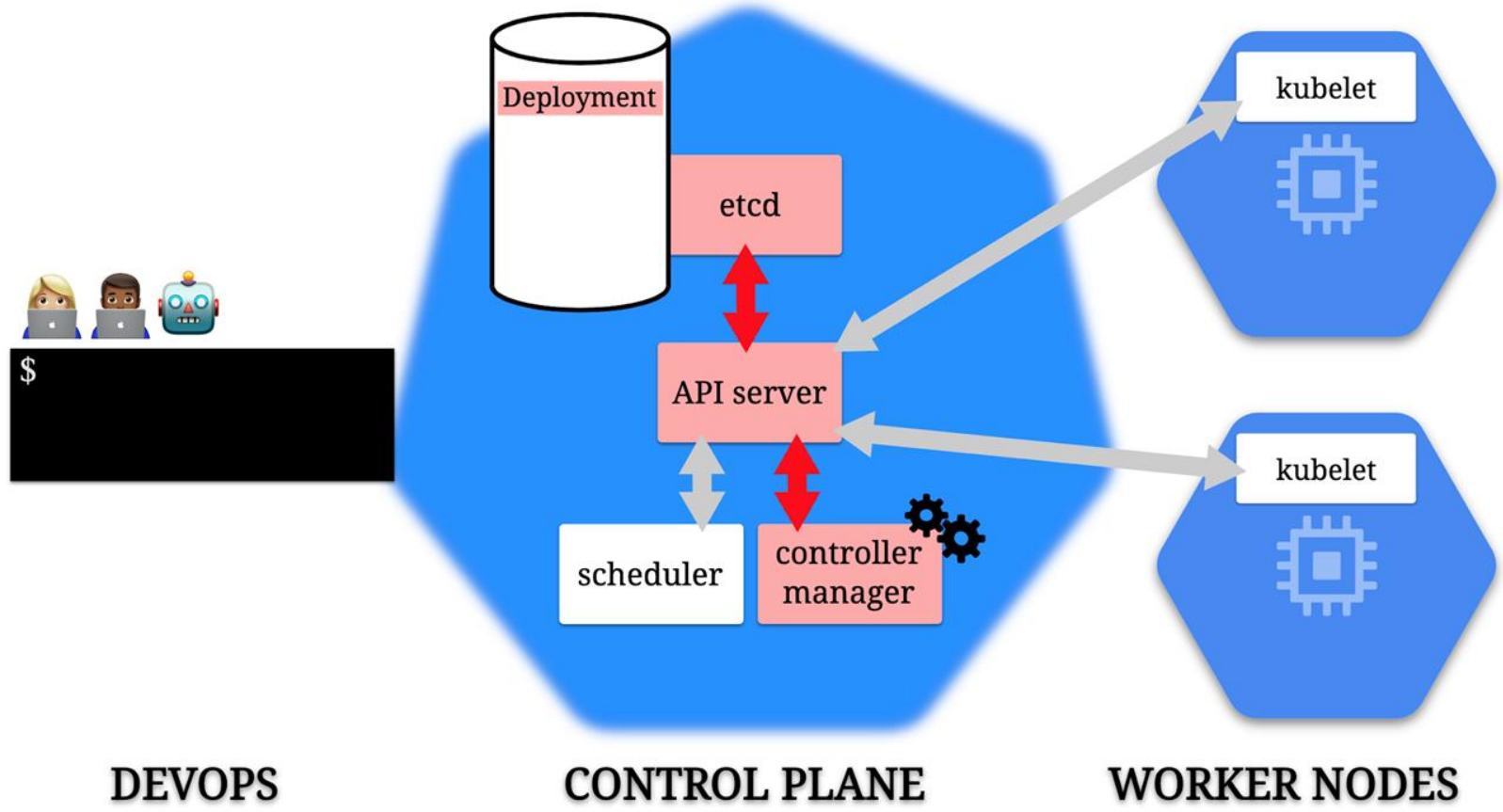

kubelet
etcd
API server
kubelet
scheduler
controller manager
$
DEVOPS
CONTROL PLANE
WORKER NODES

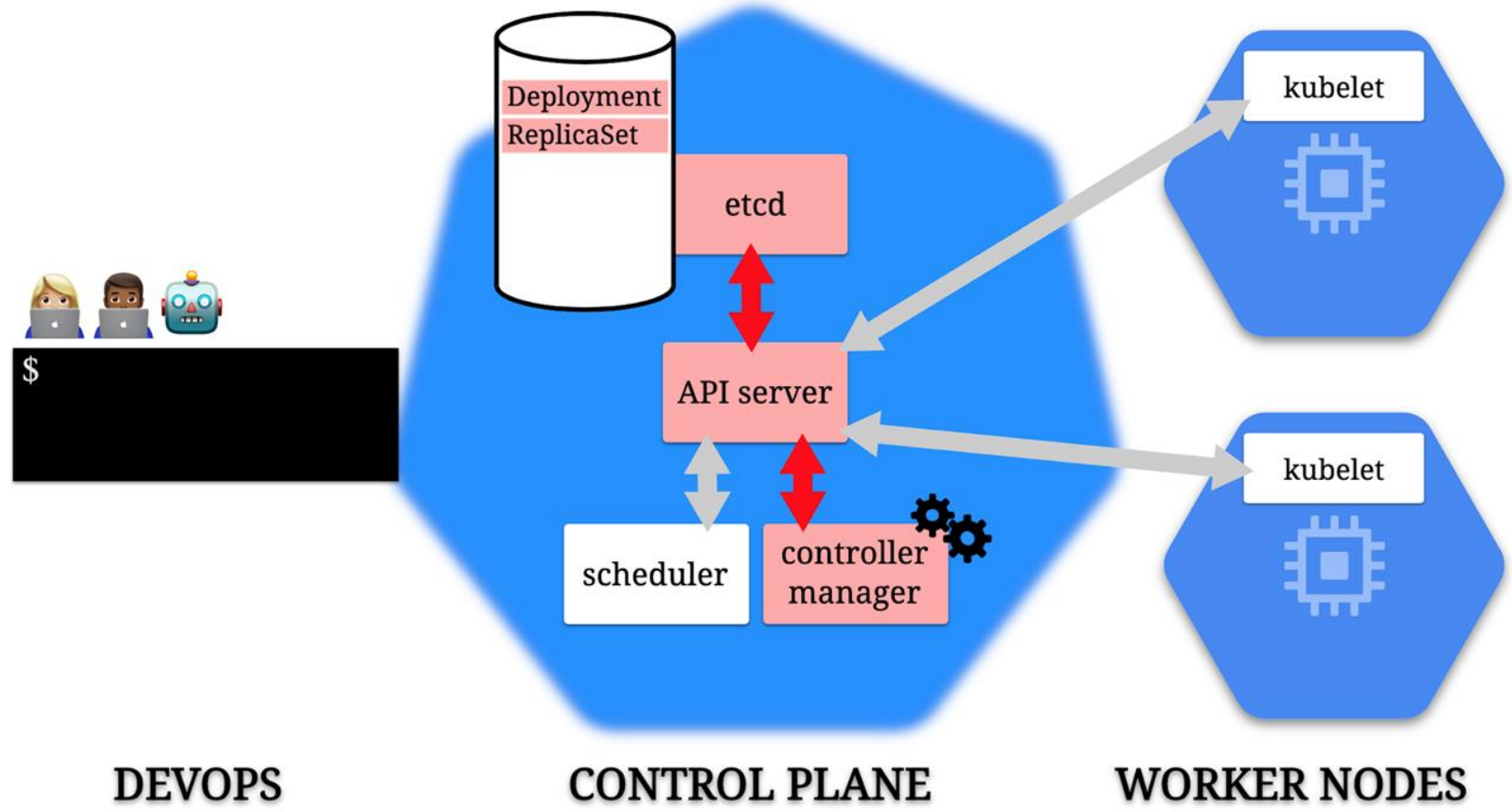

kubelet
etcd
$ kubectl run web \
--image=nginx \
--replicas=3
API server
kubelet
scheduler
controller manager
DEVOPS
CONTROL PLANE
WORKER NODES

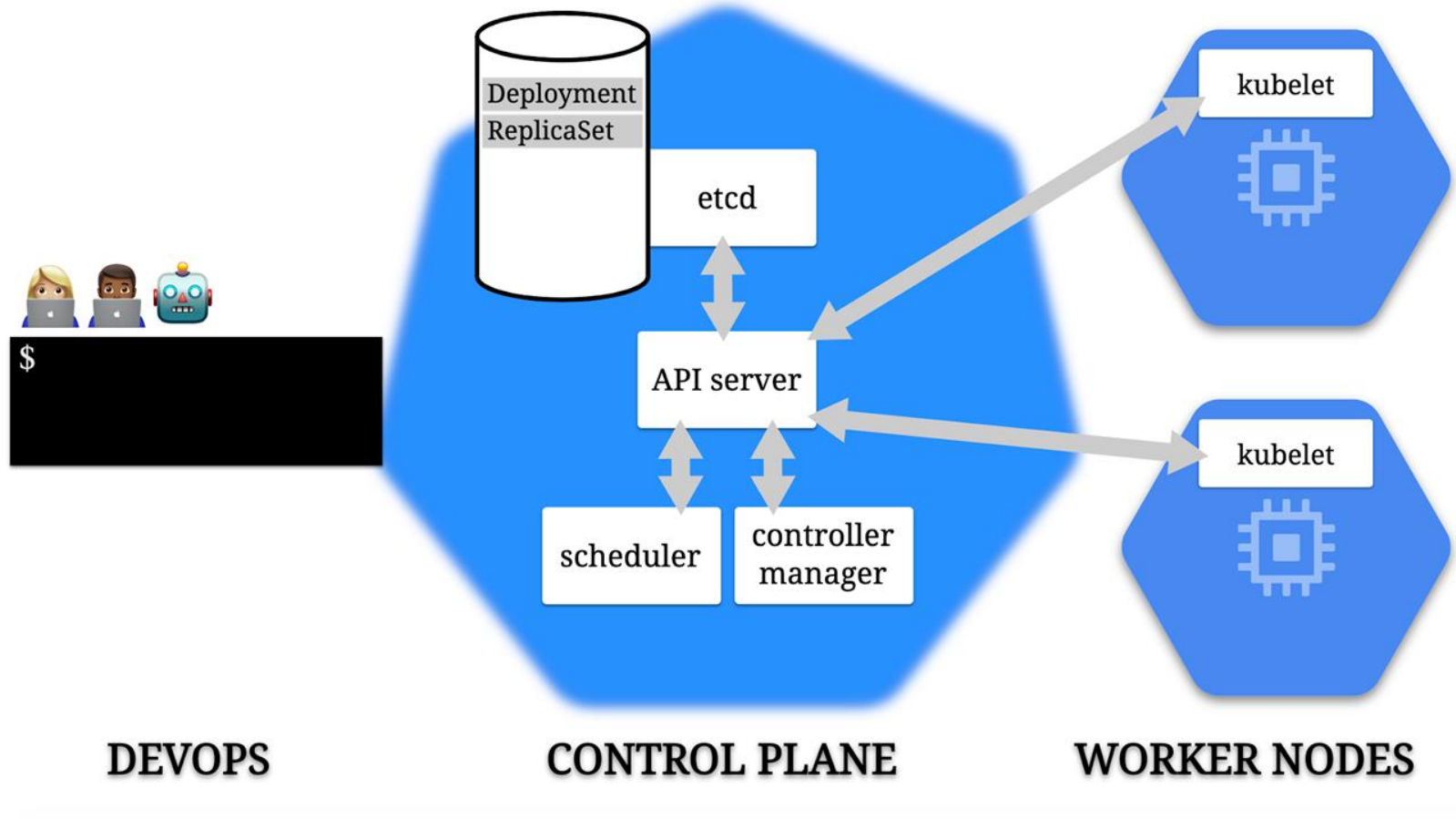

Deployment
etcd
API server
scheduler
controller manager
kubelet
kubelet
$ kubectl run web \
--image=nginx \
--replicas=3
DEVOPS
CONTROL PLANE
WORKER NODES

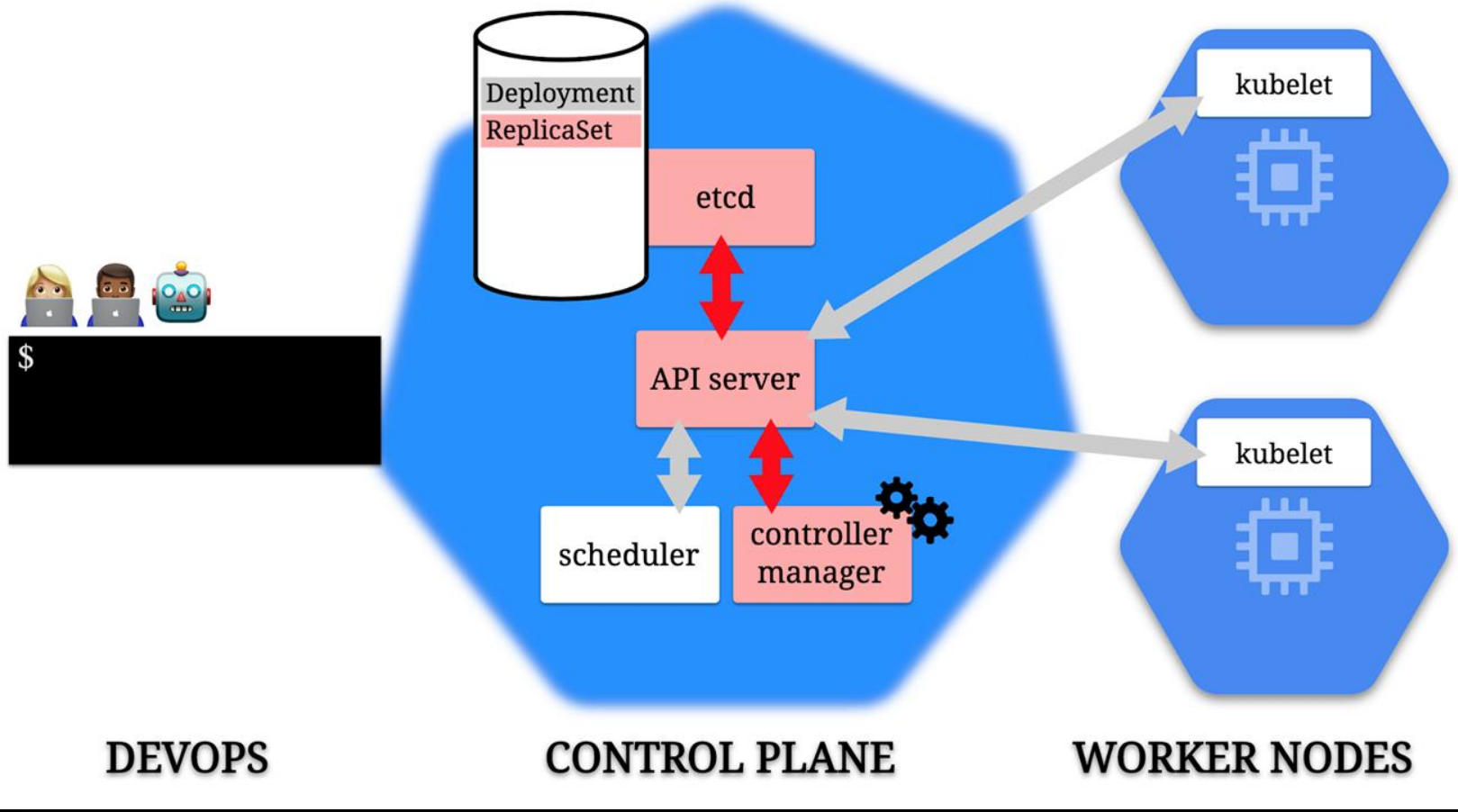

Deployment
etcd
API server
scheduler
controller manager
kubelet
kubelet
$ kubectl run web \
--image=nginx \
--replicas=3
...
deployment.apps/web
created
$
DEVOPS
CONTROL PLANE
WORKER NODES

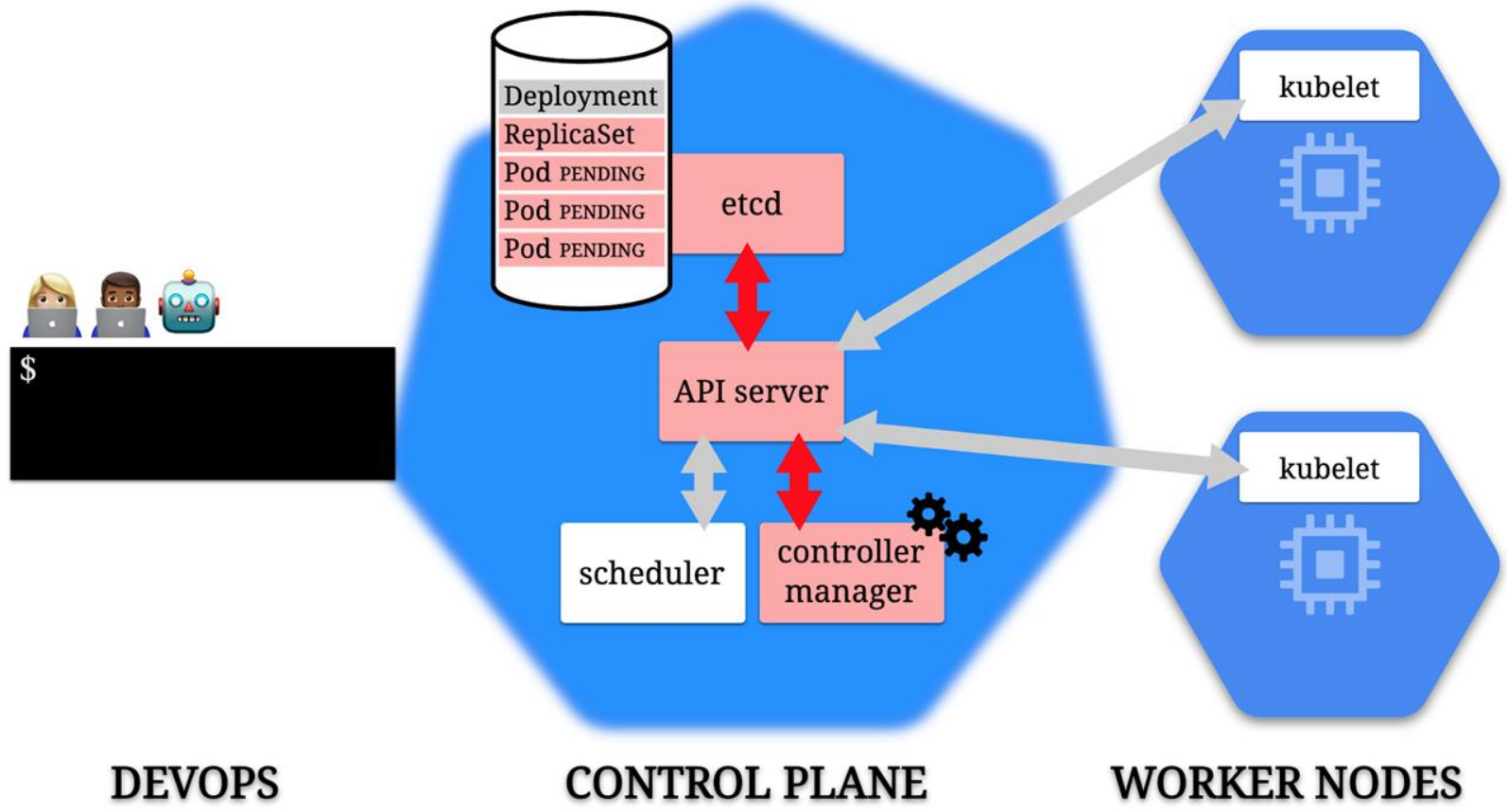

Deployment
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

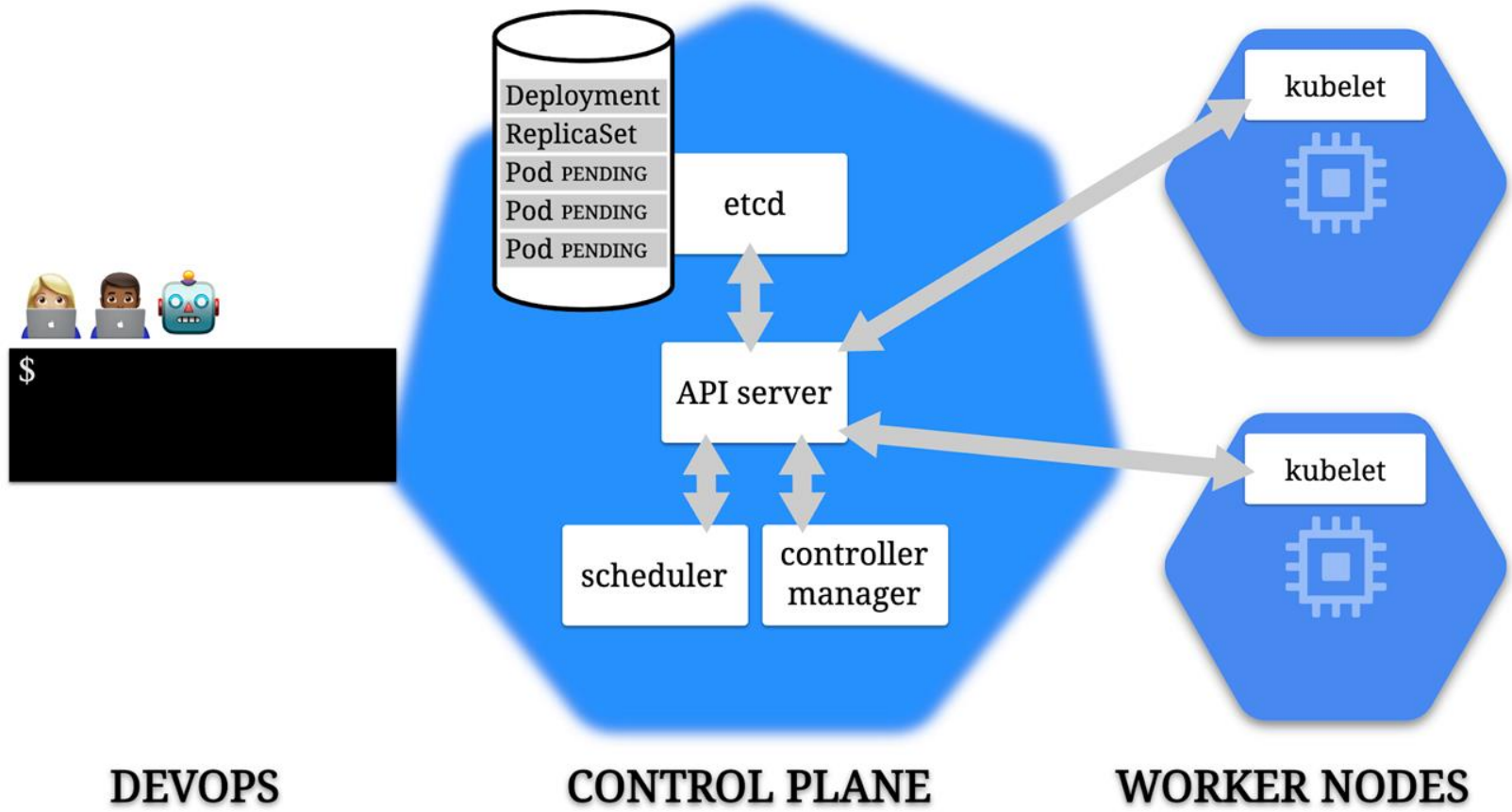

Deployment
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

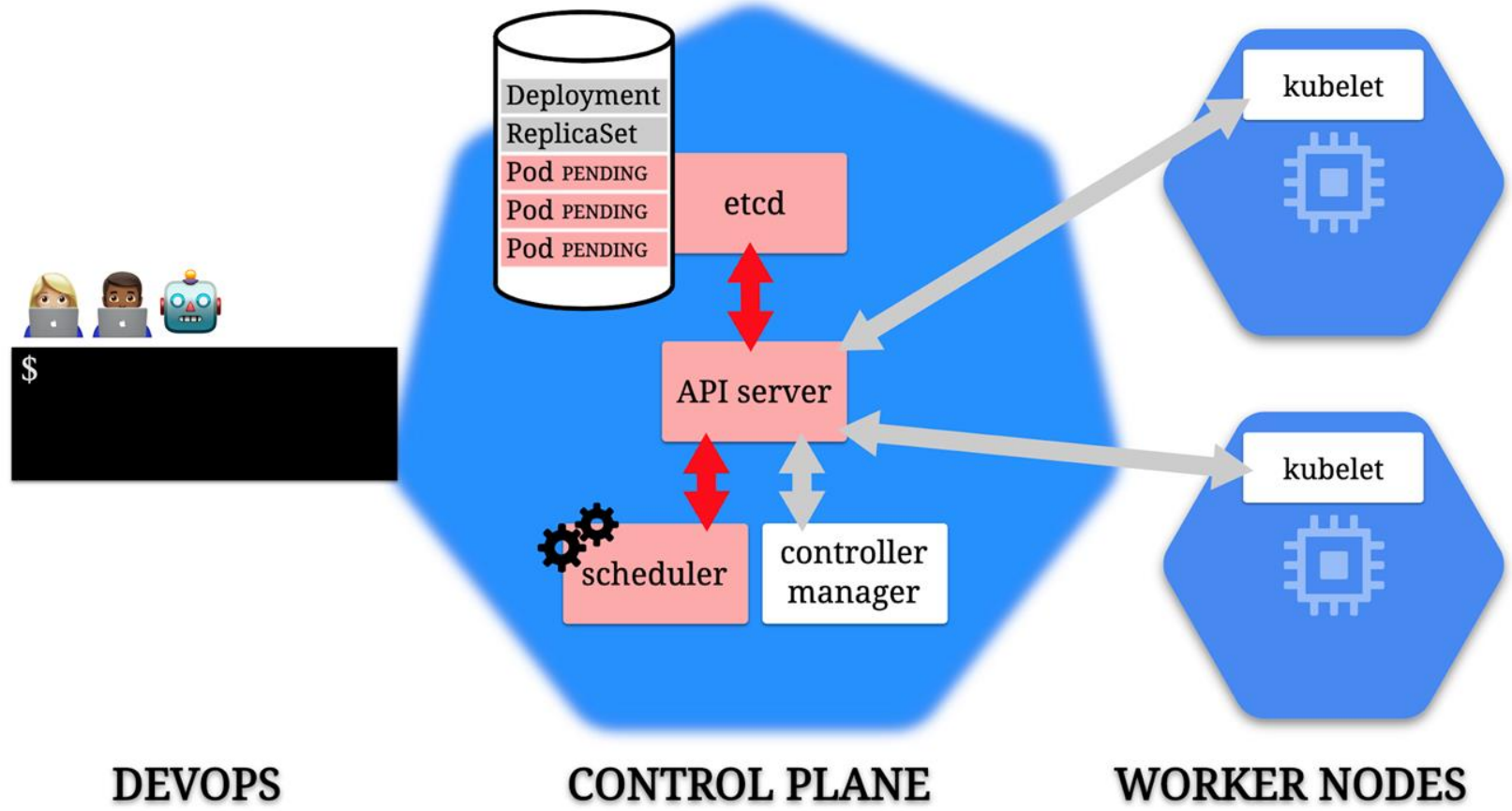

Deployment
ReplicaSet
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

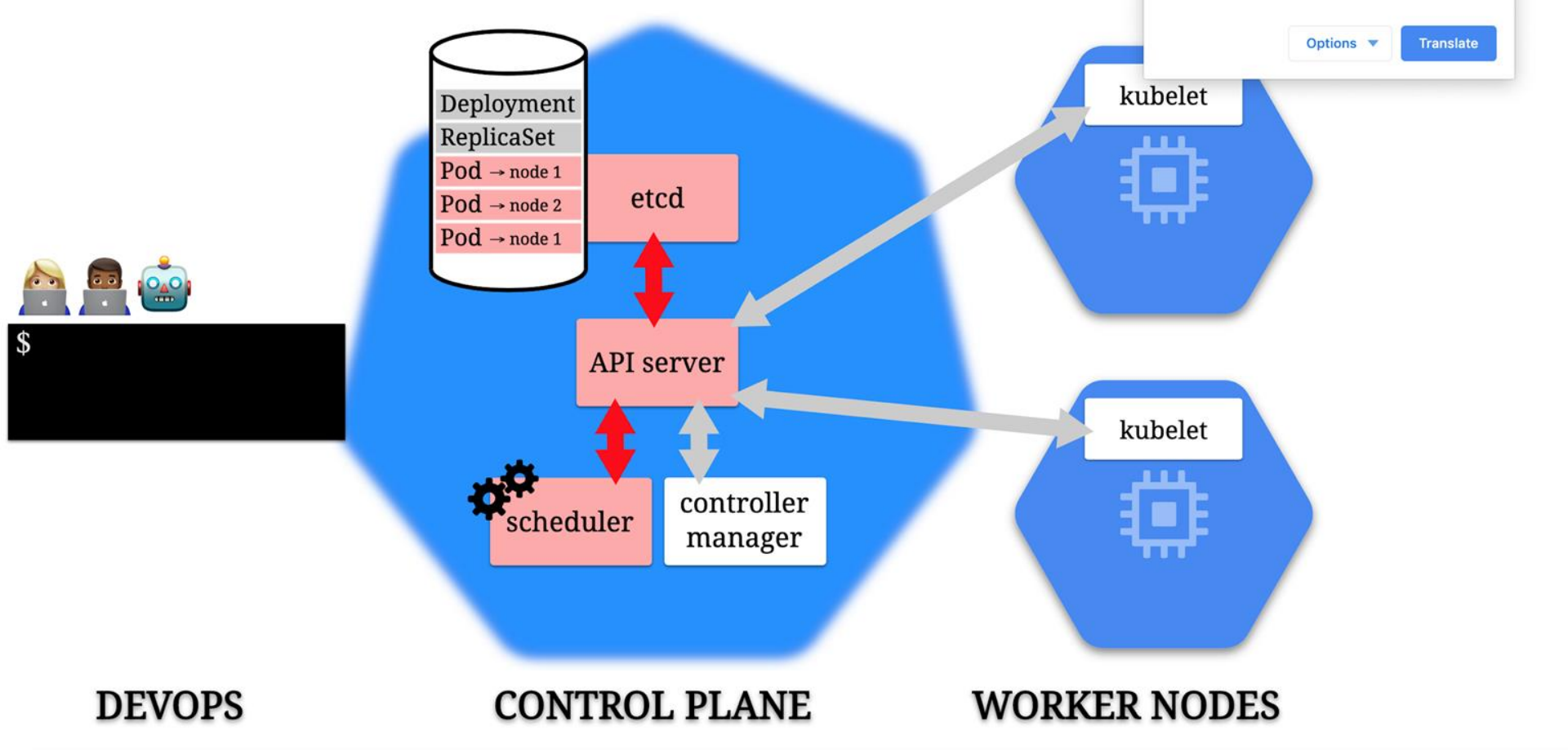

Deployment
ReplicaSet
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

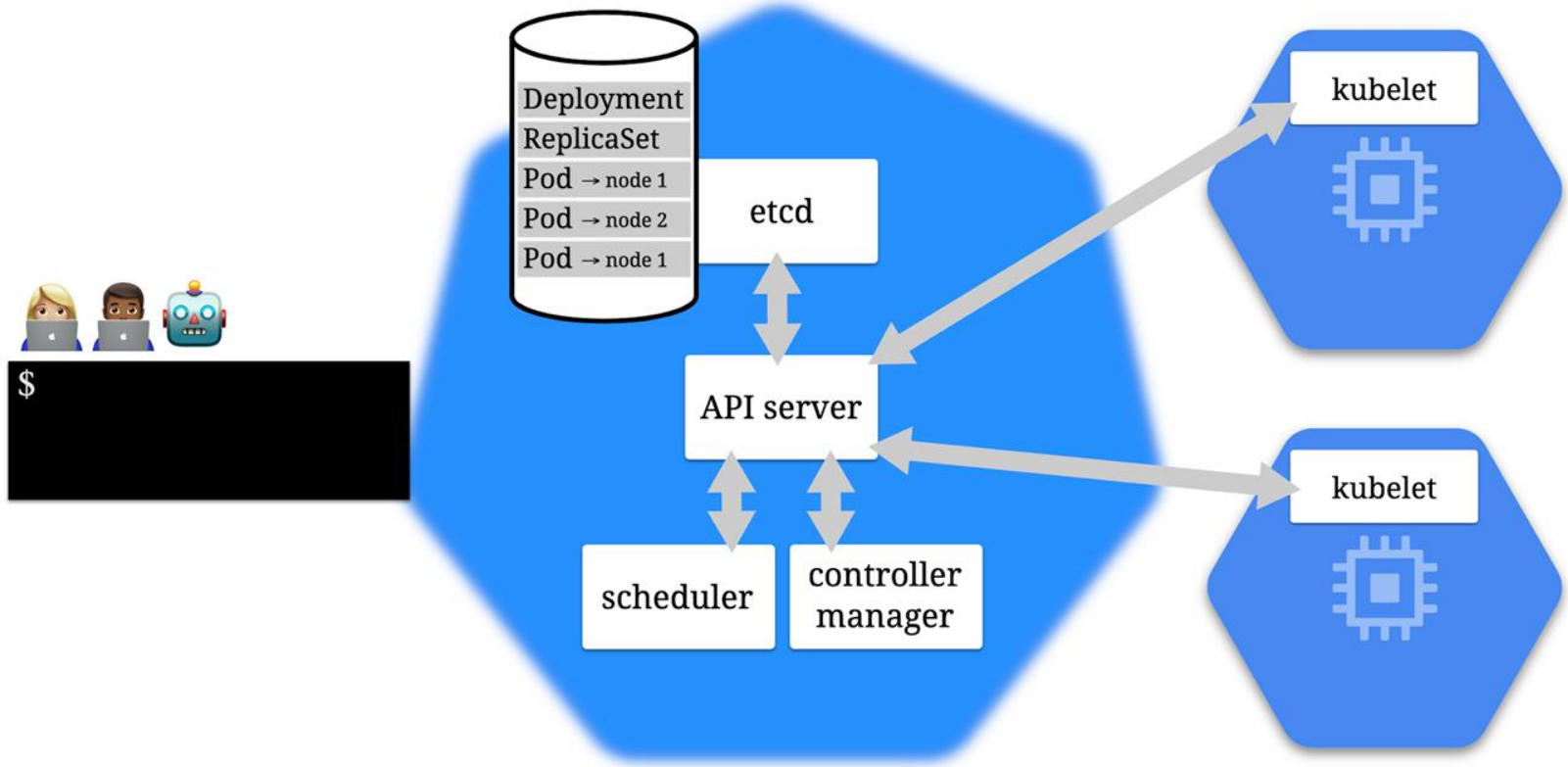

Deployment
ReplicaSet
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

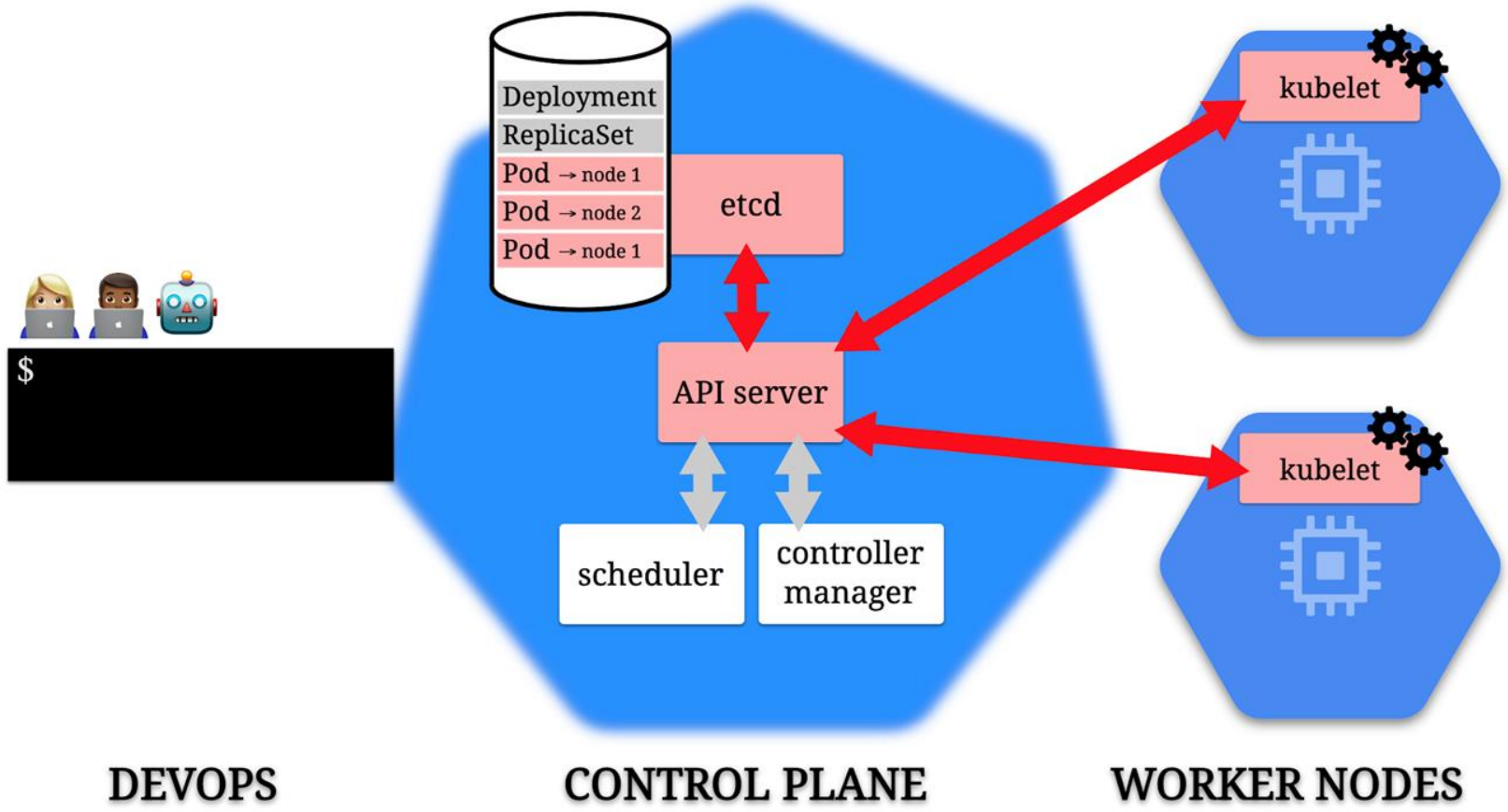

Deployment
ReplicaSet
Pod PENDING
Pod PENDING
Pod PENDING
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

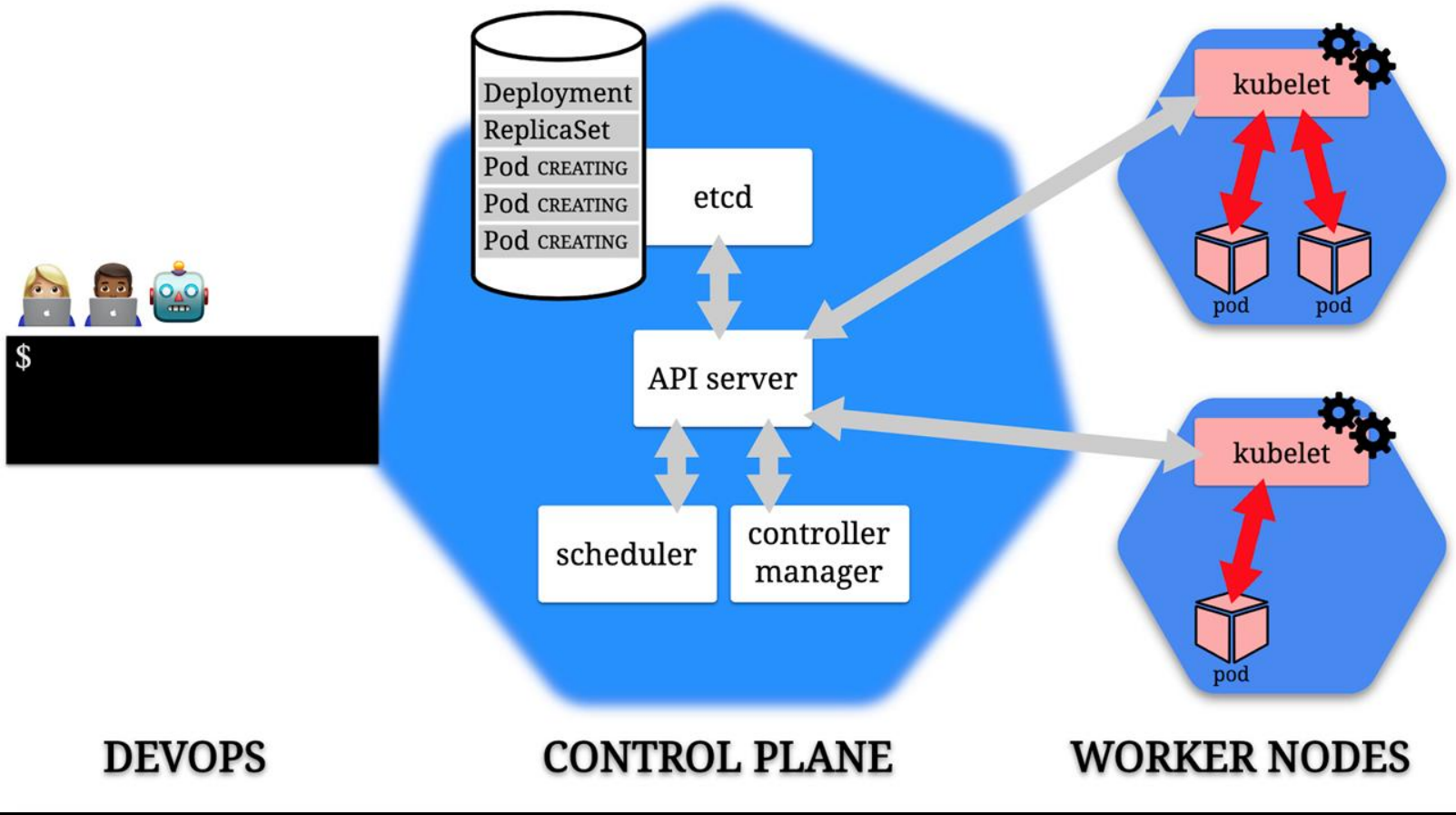

Deployment
ReplicaSet
Pod PENDING
Pod PENDING
Pod PENDING
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

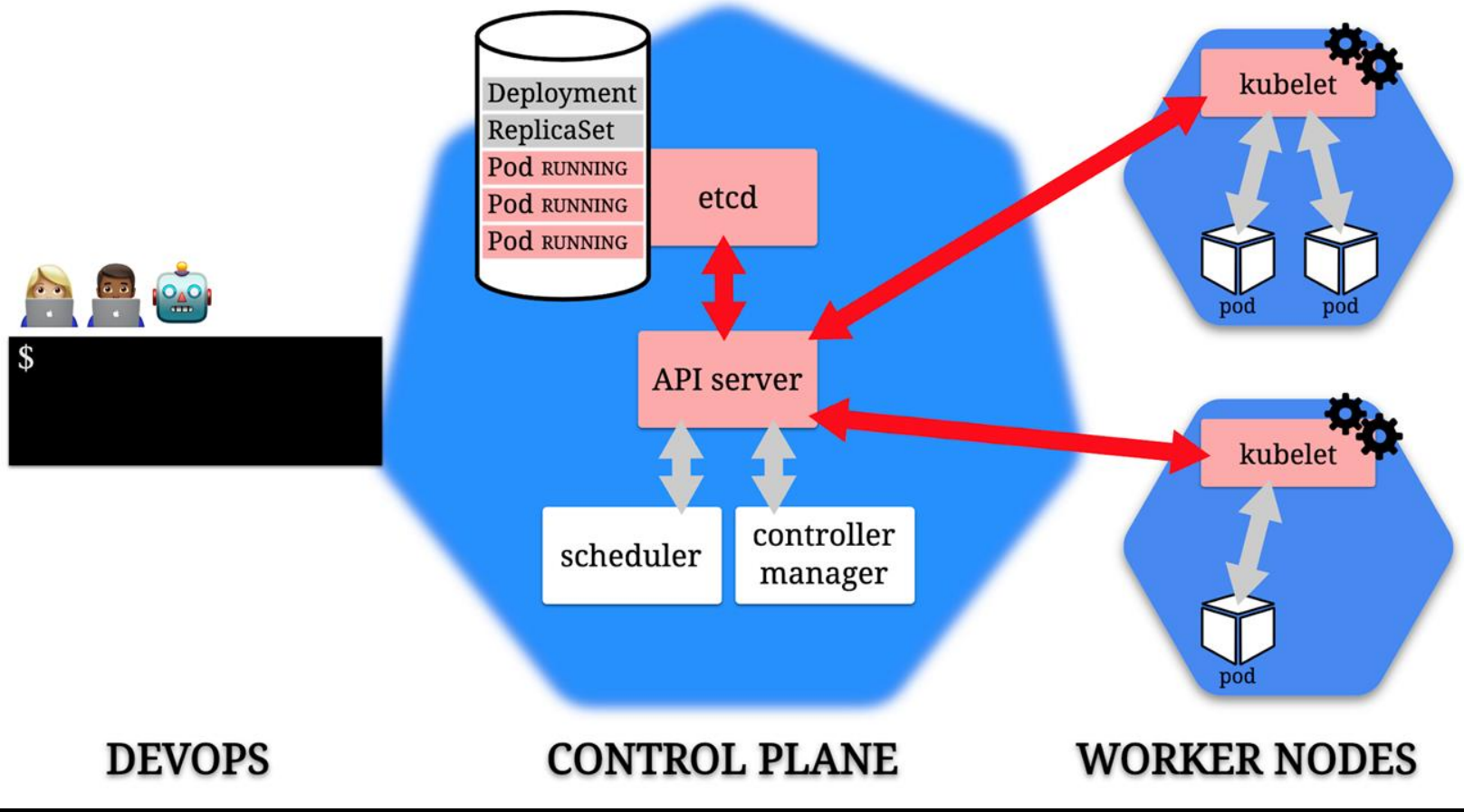

Deployment
ReplicaSet
Pod PENDING
Pod PENDING
Pod PENDING
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

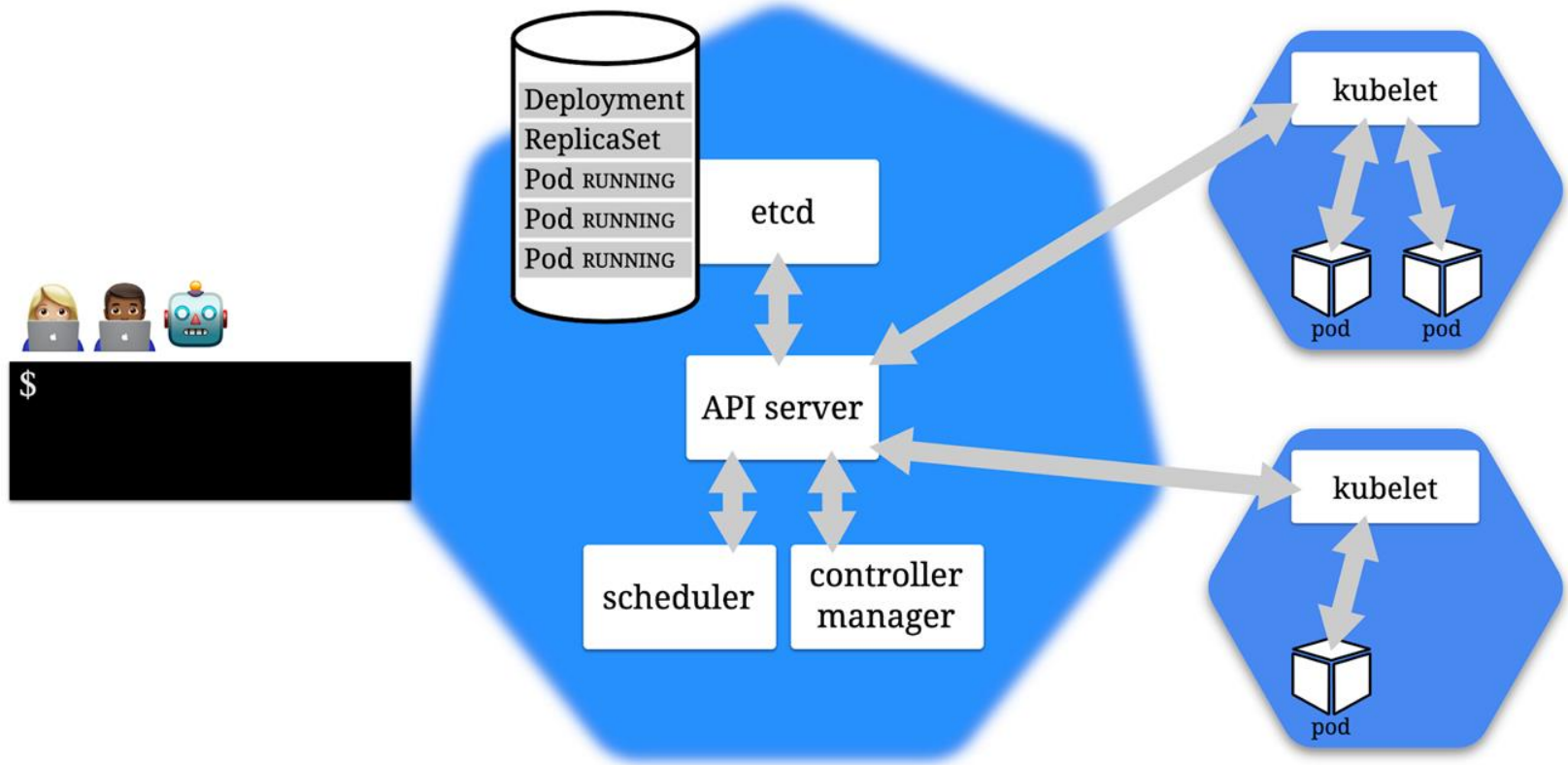

Options
Translate
Deployment
ReplicaSet
Pod → node 1
Pod → node 2
Pod → node 1
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES


Deployment
ReplicaSet
Pod → node 1
Pod → node 2
Pod → node 1
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES

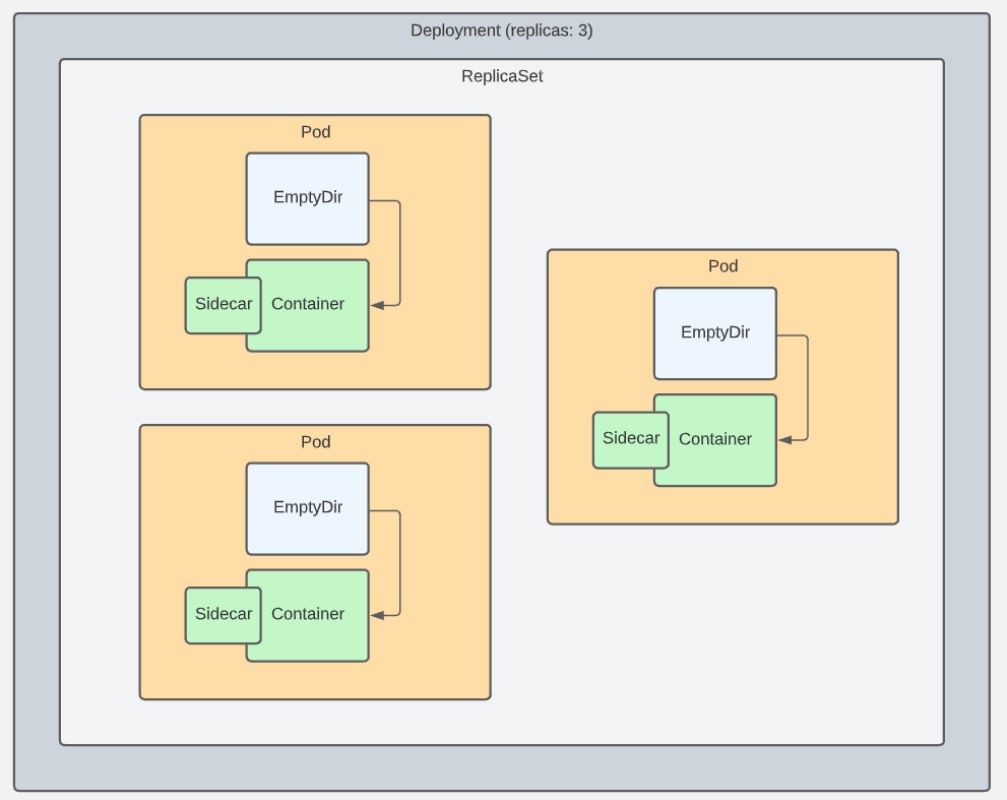

Deployment
ReplicaSet
Pod → node 1
Pod → node 2
Pod → node 1
etcd
API server
scheduler
controller manager
kubelet
kubelet
$
DEVOPS
CONTROL PLANE
WORKER NODES


Deployment
ReplicaSet
Pod CREATING
Pod CREATING
Pod CREATING
etcd
API server
scheduler
controller manager
kubelet
pod
pod
kubelet
pod
$
DEVOPS
CONTROL PLANE
WORKER NODES


Deployment
ReplicaSet
Pod RUNNING
Pod RUNNING
Pod RUNNING
etcd
API server
scheduler
controller manager
kubelet
pod
pod
kubelet
pod
$
DEVOPS
CONTROL PLANE
WORKER NODES


Deployment
ReplicaSet
Pod RUNNING
Pod RUNNING
Pod RUNNING
etcd
API server
scheduler
controller manager
kubelet
pod
pod
kubelet
pod
$
DEVOPS
CONTROL PLANE
WORKER NODES

Pods, Deployments, & ReplicaSets



Question for the Group

This component is used in a Kubernetes cluster to store state about the cluster and the resources it is responsible for:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component is used in a Kubernetes cluster to monitor current state against desired state, and notify the cluster when an update is required to resolve any gaps:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Question for the Group

This component provides the programmatic interface to the Kubernetes cluster and is used by other components and users for retrieving data about the cluster and applying changes to the cluster:

1. API server
2. Scheduler
3. etcd
4. Controllers

Type your choice in the chat…

Node Components: Container Runtime

- The software that runs the containers in the pod
- Docker is most common
- Others are containerd and CRI-O
- They need to comply with a standard called Kubernetes CRI

Addons: Cluster DNS

- A DNS system that runs in a pod in your cluster
- It also spins up a k8s Service
- Allows the use of consistent DNS names instead of IP addresses

k8s Probes

- The pods in a node are not always up and running, ready for incoming traffic
- We need a way to check
- The kubelet does this through periodic “liveness” checks into the pods in its node

kubectl

- kubectl (“kube-cuttle”) is command-line tool used to interact with API
- May require installation (automatic if using k8s via Docker Desktop)
- Config file defines details of connection to cluster (e.g., ~/.kube/config)
- Run `*kubectl cluster-info*` to confirm connectivity
- Run `*kubectl*` from command-line (with arguments) to see options
- NOTE: k8s in Docker Desktop should handle most of this for you

Lab 04

Pod Configuration

- Two types of pod: single container and multiple container
- Making a single container pod is simple – use kubectl run

$ kubectl run <name of pod> --image=<name of the image from registry>

Pod Design Patterns

- Usually, you will have one container per pod
- There are cases when multi-container pods make sense: When both pods have the same lifecycle, or they MUST run on the same node.
- An example: The primary container needs a helper process – it needs to be in the same node to be utilized.
- There are three main patterns for multi-container pods:
- Sidecar pattern
- Adapter pattern
- Ambassador pattern

Multi-container Pod: Sidecar Pattern

- Main App plus a “helper” app that isn’t part of the main app
- Example: Main app is a web app; helper is a logging utility

Multi-container Pod: Adapter Pattern

- Used when you need to standardize output form various apps in the cluster
- Each app type may format output in a unique way, but the cluster's monitoring needs to work with standardized formats
- Adding an adapter container to the pod for each app lets you still use the normal app output format in other areas, but have the cluster-level data work use a standardized format

Multi-container Pod: Ambassador Pattern

- One more way to allow the pod's containers to communicate with the outside world
- Add an "ambassador" container to the pod that acts as a proxy for network traffic to and from the primary container
- This lets you control traffic patterns based on the specific use case of the pod in question.
- Example: database connection in dev/qa/prod environments

Lab 05

Diving Deeper into Kubernetes

Namespaces in k8s

- Provide a grouping mechanism in Kubernetes
- Every k8s object belongs to a namespace
- Every Cluster automatically includes a default namespace

Namespaces in k8s

- Allow you create logical separation within a physical Cluster
- Provide a boundary for security and resource control
- Can explicitly create a namespace and deploy resources to it using namespace field in manifest or --namespace arg on kubectl

Namespaces in k8s

- Objects within a namespace are isolated from others
- Enables creation of multiple instances of same apps with same names
- Resources from one namespace can communicate with another namespace using Services
- Controller only looks for matching resources in its namespace

Contexts

- Can be used to define connection details for a k8s Cluster
- Sets the default namespace to be used
- Prevents need to specify namespace – will use context to automatically set if excluded (like with default in standard install)
- Able to switch between contexts using use-context command

Contexts

- Can also be used to manage switching between different Clusters (local or remote)
- For remote, API server will be secured with TLS and kubectl uses a client cert for AuthN

ConfigMaps vs. Environment Variables

- A ConfigMap is an API object that lets you store configuration for other objects to use
- These are key/value pairs
- This lets you decouple environment-specific config data from your container images, making your apps more portable
- CAUTION: ConfigMaps do not provide encryption or secrecy. Use a Kubernetes Secret for that.

ConfigMaps

- Can be:
  - Set of key-value pairs
  - Blurb of text
  - Binary files
- One pod can use many ConfigMaps and one ConfigMap can be used by many pods
- Data is read-only – pod can’t alter

ConfigMaps as Env Vars

- Can be created from a literal:
  `*kubectl create configmap <name> --from-literal=<key>=<value>*`
- Can be created from a file (to group together multiple settings):
  `*kubectl create configmap <name> --from-env-file=<file-path>*`

Where <file-path> contains .env file like:

key1=value1
key2=value2

- Can be created from a .yml definition file (like all things k8s)

ConfigMaps

- Can be presented as files inside directories in the container
- Uses volumes – making contents of ConfigMap available to pod
- Uses volume mounts – loads contents of ConfigMap volume into specific container path in pod
- Can contain settings to override defaults

ConfigMaps - Precedence

- If env vars defined in multiple places, definitions in `env` section in pod spec will trump others (localized)
- Typical approach:
  - Default app settings “baked in” to container image (e.g., to support dev mode)
  - Specific settings for an environment stored in ConfigMap and surfaced to container filesystem
  - Merges with default settings (overwriting where applicable)
  - Final tweaks can be accomplished with env variables in pod spec

Secrets in Pods

- Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- A pod needs to reference a secret. There are three ways get to the secret:
- As a file in a Volume that is mounted to a container;
- As an environment variable for a container;
- By the kubelet, when it pulls images for the pod
- NOTE: The default config for a Secret does NOT have encryption – data is plain text. You need to configure “Encryption at Rest” for the Secret”, and Role Based Access Control in your cluster

Lab 06

Lab 07

k8s Services

- Services are an abstraction
- They define a set of pods, and an access policy for them
- A pod has an IP address
- Pods are ephemeral – what happens to IP traffic if they die?
- Pods in a service can have fixed IP addresses that stay the same even if the actual pod dies and is spun up again
- Services are k8s objects; they are created like any other

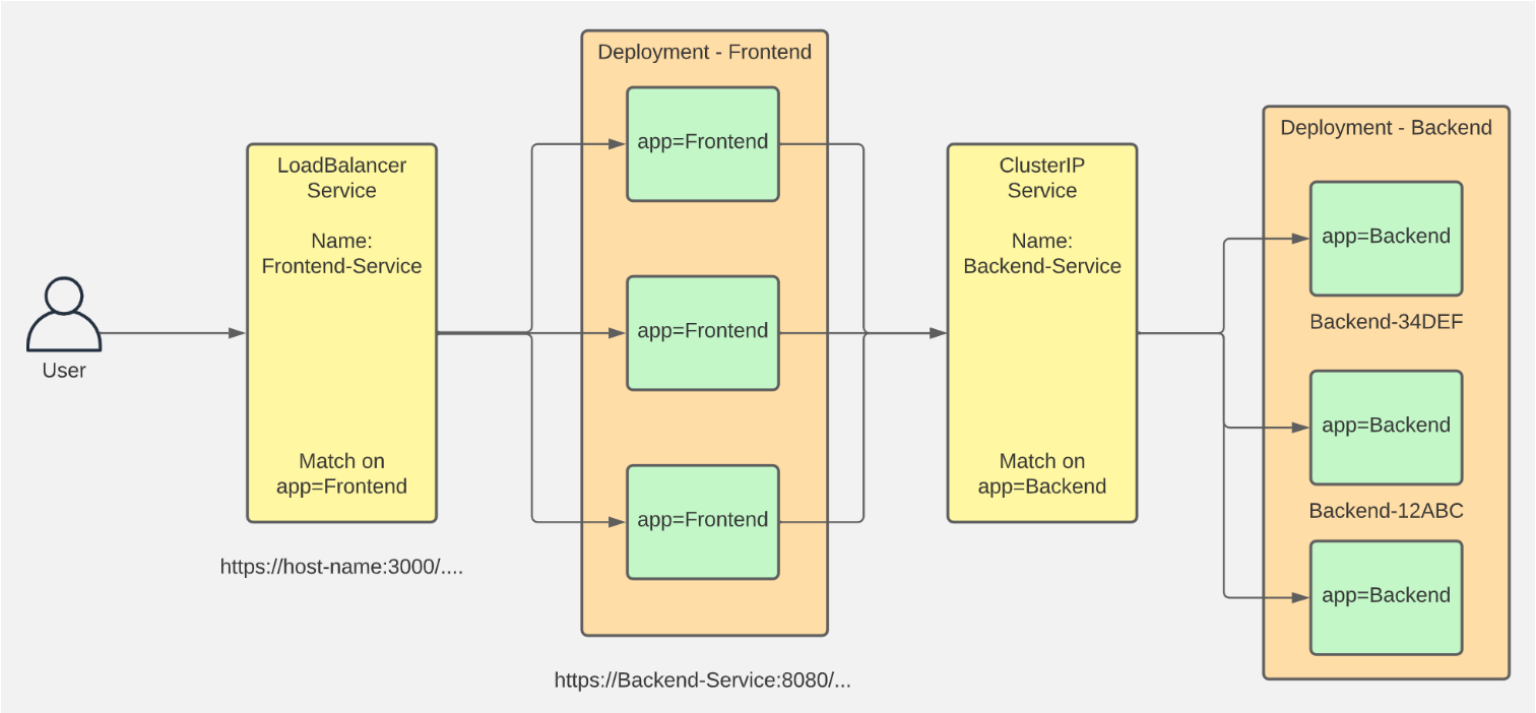
k8s Service Types

- ClusterIP (default): IP is internal to the cluster
- NodePort: IP for the port is exposed with a static IP address
- LoadBalancer: IP traffic to nodes is managed by external load balancer
- ExternalName: IP traffic is routed by DNS Name (foo.example.com)

Cluster Access – Internal vs. External

- You can access the cluster API using kubectl (“proxy”)
- You can get libraries for popular languages that can access the cluster
- k8s objects (pods) can also access the API internally
- This uses an internal DNS
- “kubernetes.default.svc” maps to the API server
- The pod usually uses a “sidecar” container to do this

Services


Deployment - Frontend
app=Frontend
app=Frontend
app=Frontend
LoadBalancer
Service
Name:
Frontend-Service
Match on
app=Frontend
User
https://host-name:3000/....
ClusterIP
Service
Name:
Backend-Service
Match on
app=Backend
Deployment - Backend
app=Backend
Backend-34DEF
app=Backend
Backend-12ABC
app=Backend
https://Backend-Service:8080/...

Lab 08

Lab 09

Lab 10

Container File System

- Each container in a pod has its own filesystem built by k8s
- Can be built from multiple sources, including:
  - Layers from the container image
  - Writable layer for the container
- Can be expanded with other sources like ConfigMaps and Secrets – mounted to a specific path in a container
- Volumes can provide another source of storage – volumes are defined and mounted to containers

EmptyDir

- Empty directory stored at pod level vs. container level
- Mounted as a volume into the container so visible there
- Any data stored there from the container remains in pod between restarts
- Replacement containers can access data from predecessors

HostPath

- Data physically stored on node in cluster
- Extends beyond lifecycle of pod
- Available to replacement pods (but only if run on same node)
- Mounted as a volume into the container like others
- However, can have issues:
  - In cluster with multiple nodes, limits usefulness
  - If not careful, can expose large blocks of host data

PersistentVolume

- Like pods (abstraction over computer) and services (abstraction over network), persistent volumes provide abstraction over storage
- k8s object that defines available section of storage
- Can be “mapped” to shared storage (like NFS) or locally (for dev/test purposes)

PersistentVolumeClaim

- Pods are not allowed to use persistent volumes directly
- Instead, pods claim using PersistentVolumeClaim object type in k8s
- Requests storage for a pod
- k8s handles matching up a claim to a persistent volume
- Provides virtual storage abstracted from actual storage

Dynamic Provisioning

- Static provisioning accomplished by explicitly creating persistent volume and then claim (k8s binds claim to volume)
- Dynamic provisioning supported as well
- You create the persistent volume claim and let persistent volume be created on demand by cluster
- Can leverage storage classes for defining and matching types of storage

Storage Classes

Defined by three properties:

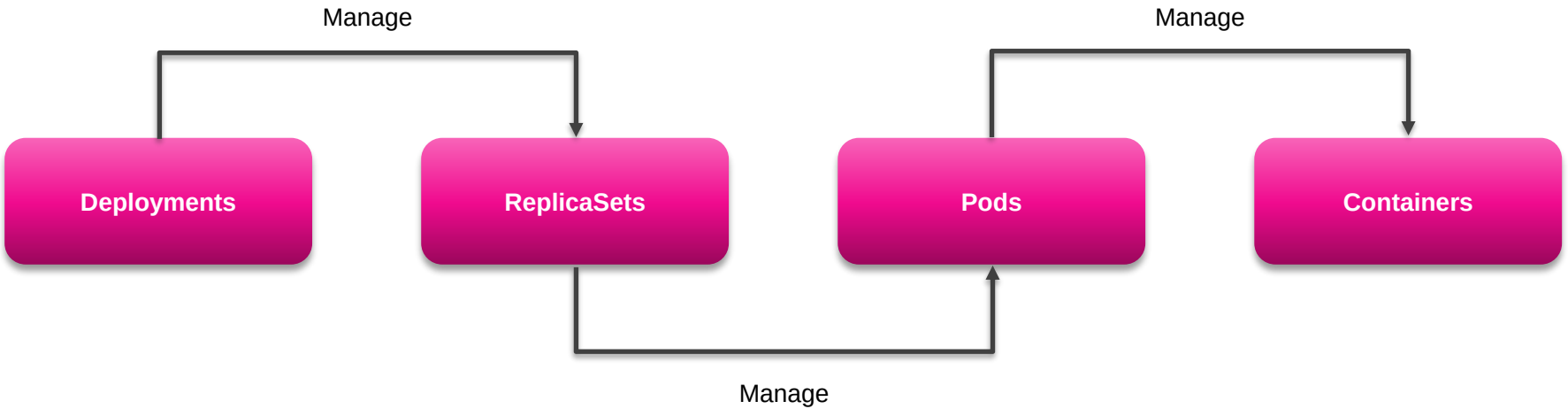
- provisioner – component that creates persistent volumes on demand
- reclaimPolicy – what to do with dynamically created volumes when claim is deleted
- volumeBindingMode – eager vs. lazy binding/creation (at same time as claim creation or only when pod using the claim get created)

Also, able to create custom storage classes defined by same properties

Lab 11

Demo – Deploying WordPress and MySQL

Scaling k8s


Manage
Manage
Deployments
ReplicaSets
Pods
Containers
Manage

ReplicaSets

- Can be created directly as an object in k8s
- However, better practice would be to create/manage through deployments
- `replicas` property in pod spec in deployment can be used to build out multiple replicas for a pod config
- If not included, defaults to 1

Setting Limits in Deployments

- When defining pod spec in deployment, can use limits to manage
- `resources` attribute defines compute resource requirements for pod
- `limits` can be used to apply limits (e.g., in amount of memory and cpu)
- Usually better to optimize once running vs. prematurely optimizing
- Target environment when running will help determine correct settings

DaemonSets

- daemon is usually a system process that runs constantly as single instance
- In k8s, DaemonSet runs single replica of a specific pod on every node in cluster
- Usually used for infrastructure-level concerns:
  - Logging/central data collection
  - High availability for a reverse proxy running in cluster
- Looks similar to other controllers (e.g., Deployment) but no replica count

Using DaemonSets in Deployments

- Control loop watches for nodes joining cluster
- New pod instance for the daemon will be started on new node
- Works to stop/remove daemon pod(s) if node(s) taken out of cluster

Using DaemonSets in Deployments

- Can also target just a subset of nodes for daemon pod
- As with most other things in k8s, use label matching
- Watches for nodes join cluster and for metadata matching label
- DaemonSet control loop is different from ReplicaSets because has to monitor combo of node activity and pod count

Lab 12

Jobs

- Provide options for running maintenance tasks in your k8s Cluster
- For example, with PostgreSQL cluster from previous, can schedule tasks against the DB “out-of-band”
- Tasks like backing up databases, data migration, data loads, etc.
- Defined with a Pod spec and run to completion as a batch process

Jobs

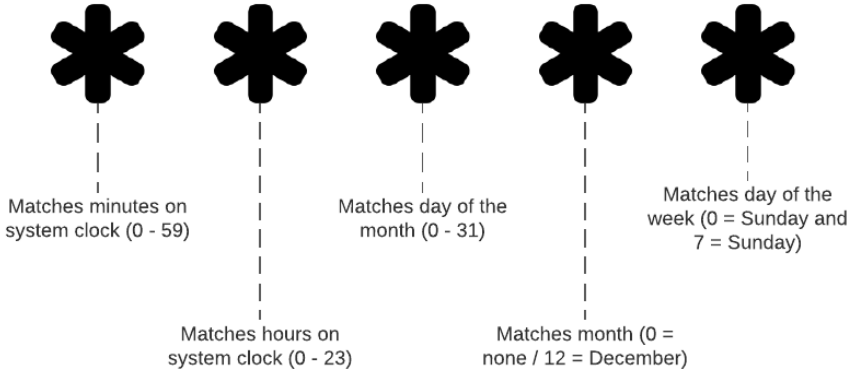
- Can run Pods for any container image but should represent a process with a discrete end (intended to run to completion)
- Jobs add labels to the underlying Pods created
- Includes a completions property for specifying how many times the Job should run

Jobs

- And a parallelism property for specifying how many Pods to run in parallel
- Enables distribution of Job processing for multiple types/instances across the Cluster
- Also, can assist with managing the speed of Job execution

CronJobs

- Specific type of Job for executing regularly and on a schedule
- Allows definition of a Job spec for use by CronJob for Job creation on the defined schedule
- Use the Linux Cron format



CronJobs

- Can move CronJobs to suspended state
- Logic needs to account for standard Pod lifecycle management
- Could include restarts or parallel runs – logic needs to support

Lab 13

Lab 14

Startup Probes

- First probe executed on startup and determines if app is initialized and ready for traffic
- Only runs on initialization – once complete, will not run again in the Pod
- Can specify number of probe attempts (failureThreshold)
- And number of seconds to wait between retries (periodSeconds)
- If all defined attempts fail, k8s will kill the Pod and startup a replacement

Readiness Probes

- k8s knows if Pod container is running but not if application inside is healthy (specific to the app)
- Defined in the Pod spec
- Execute health checks on a fixed schedule

Readiness Probes

- Act at the network level and manage routing for listening components
- If deemed unhealthy, Pod will be taken out of ready state and removed from list of active Pods
- k8s keeps running probe looking for Pods that have recovered for reassociation to the Service
- Different types of probes are supported – e.g., HTTP GET for web apps

Liveness Probes

- Use same health checks as readiness probes
- Action on failure is different
- Act at compute level and restart unhealthy Pods
- Different types of probes are supported – e.g., HTTP GET for web apps

Lab 15

Managing Logs

- k8s collects logs from container runtime and stores as files on node
- Goal is to get logs out of the container and potentially centralize for aggregated processing
- Often uses a sidecar container for the aggregation

Managing Logs

- Logs can benefit from normalization
- Parsing can be applied as the information passes through the pipeline
- Several 3rd party tools (e.g., Fluentd for log collection, Elasticsearch for log storage, and Kibana for view)

Deploying and Scaling Clusters

- Clusters can scale as needed
  - Increased load
  - Rolling cluster updates
- This is often managed by the cloud provider with cloud native

Managing Cluster Resources

- Container limits provide options for specifying limits at the container level
- Allow specification of memory and cpu limits to prevent a container from exhausting cluster resources
- Can be defined using “requests” and “limits” in container spec
- “requests” define what is requested (like a minimum) – container can use more than what’s defined if node provides
- “limits” are hard stops – if exceeded container will be replaced

ResourceQuota

- Also, able to apply resource limits at the namespace level
- Pod specs need to include a resource section
- Container limits are reactive; ResourceQuotas are proactive
- Pods won’t be created if they would exceed limit

The Ingress Controller

- Specialized load balancer for k8s environments
- Manages traffic to pods in your cluster
- Like Nodes, Services etc., they are deployed using the cluster API
- There are several available besides the k8s offering

Lab 16

Rolling Deployments

- Code changes, so your pod (containers) change
- We want a stable experience for users
- k8s does this via rolling deployments
- As updates nodes become available, traffic is load balanced to ensure uptime

Blue/Green Deployments

- How they work: keep two identical environments, conventionally called blue and green, to do continuous, risk-free updates. This way, users access one while the other receives updates.
- With bare metal servers, this is expensive – one of the environments isn’t used much when the other is
- k8s makes this easy – we can make a second environment, switch traffic over, then kill the first with little impact

Canary Pods for Testing and Production

- A strategy to try out a new feature only for a small segment of users or traffic (A/B testing)
- k8s makes this easy
- Use a tag to designate what pods will receive a small percentage of traffic
- Also, allows smoke testing on small scale before “opening floodgates”

NetworkPolicy and SecurityContext

- NetworkPolicy can be used to manage access within a Cluster
- Work like firewall rules and use label selectors for blocking traffic
- Provides ingress rules and egress rules to manage flow
- Rely on network implementation in Cluster for enforcement

NetworkPolicy and SecurityContext

- SecurityContext provides an option for container security
- Alternative to containers running as root or super user account
- Offers more fine-grained control than at Pod level, enabling explicit addition and removal of Linux kernel options
- Must remain aware of potential breaks in apps caused by security config

Lab 17

Lab 18

Demo – Blue/Green Deployments

Helm

Helm

- No real concept of “applications” in k8s
- Each “application” is a collection of YAML files that define the required resources
- Helm provides the concept of a repository in which the set of YAML files for an “application” can be stored and pulled/executed together
- Supports concept of public and private repositories
- Separate install → https://helm.sh/docs/intro/install/

Helm

- To use a repository, add using *helm repo add <local name> <url>*
- Update local repository cache using *helm repo update*
- Use *helm search repo <app name> --versions* to search for app in the repository cache

Helm

- Client-side tool that uses same connection info as Kubernetes
- Extends YAML to provide support for things like parameters
- An “application” package is a *chart*
- Charts can be developed/deployed locally or published to a repository
- Installing a chart creates a *release*
- Each release has a name – can install multiple instances of same app by using different names

Helm

- Manifests in a chart can support parameters
- Allows varying specific settings at time of install
- Each chart contains a set of default values

Helm

- Use *helm show values* to see defined default values for parameters
- Use *helm install* to install a chart
- Use *--set* flags on *install* to change parameters from defaults
- *helm ls* will show you installed releases

Helm

- Can create your own charts
- *helm create* scaffolds a boilerplate structure
- Provides Chart.yaml, values.yaml, templates folder
- Use {{ }} to templatize your manifest with a parameter
- Can use *helm lint* to validate contents of chart

Role-Based Access Control (RBAC)

RBAC (Role-Based Access Control)

- System of “least privilege”
- Allow only, no deny – absence of a permission implies denial
- RBAC is used to grant permissions on resources in k8s
- All resources can be secured
- Permissions get applied to a subject (user, service account, group) but not directly

RBAC (Role-Based Access Control)

- Set permissions in a role
- Apply a role to one or subjects using a role binding
- Role and RoleBinding operate at the level of namespace
- ClusterRole and ClusterRoleBinding operate at the cluster level

RBAC (Role-Based Access Control)

- Kubernetes does not authenticate end users – delegates to external identity providers
- Able to connect to your corporate provider (for example, LDAP or Active Directory)
- Cloud k8s offerings integrate with their available provider
- Can configure custom OIDC (Open ID connect) provider but not a simple task
- If using a managed k8s service, authentication will be built in

RBAC (Role-Based Access Control)

- One option to create a “user” within Kubernetes is via certificates
- Kubernetes can issue client certificates for end users requested with a username
- Certificate supplied on Kubernetes API requests is used to validate permissions to accomplish a task – attached by name
- Permissions are all additive – start with none (“least privilege”)
- Able to impersonate a user using the *--as* flag on a kubectl command

RBAC (Role-Based Access Control)

- Common approach is to start with predefined cluster roles and bind to one or more subjects for a specific namespace
- Role bindings provide a layer of abstraction between users and the roles they have

Demo – Bringing it All Together

Thank you!

If you have additional questions, please reach out to me at: asanders@gamuttechnologysvcs.com

DevelopIntelligence
A PLURALSIGHT COMPANY