WA2919 Booz Allen Hamilton Tech Excellence Cloud Engineering Program - Phase 3

For customizations of this book or other sales inquiries, please contact us at:

USA: 1-877-517-6540, email: getinfousa@webagesolutions.com
Canada: 1-877-812-8887 toll free, email: getinfo@webagesolutions.com

Table of Contents

Chapter 1 - Docker Introduction

Objectives

Key objectives of this chapter

- Docker introduction
- Docker Engine Architecture
- Docker command-line

1.1 What is Docker

- Docker is an open-source (and 100% free) project for IT automation
- You can view Docker as a system or a platform for creating virtual environments which are extremely lightweight virtual machines
- Docker allows developers and system administrators to quickly assemble, test, and deploy applications and their dependencies inside Linux containers supporting the multi-tenancy deployment model on a single host
- Docker's lightweight containers lend themselves to rapid scaling up and down
  - *Note:* A container is a group of controlled processes associated with a separate tenant executed in isolation from other tenants
- Written in the Go programming language

1.2 Where Can I Run Docker?

- Docker runs on any modern-kernel 64-bit Linux distributions
- The minimum supported kernel version is 3.10
  - Kernels older than 3.10 lack some of the features required by Docker containers
- You can installl Docker on Linux, Windows, or macOS.
- Docker can be installed on Windows using Docker Desktop and Windows Subsystem for Linux (WSL2).
  - Hyper-V is required depending on the Windows OS edition you use.

1.3 Installing Docker Container Engine

- Installing on Linux:
  - Docker is usually available via the package manager of the distributions
  - For example, on Ubuntu and derivatives:

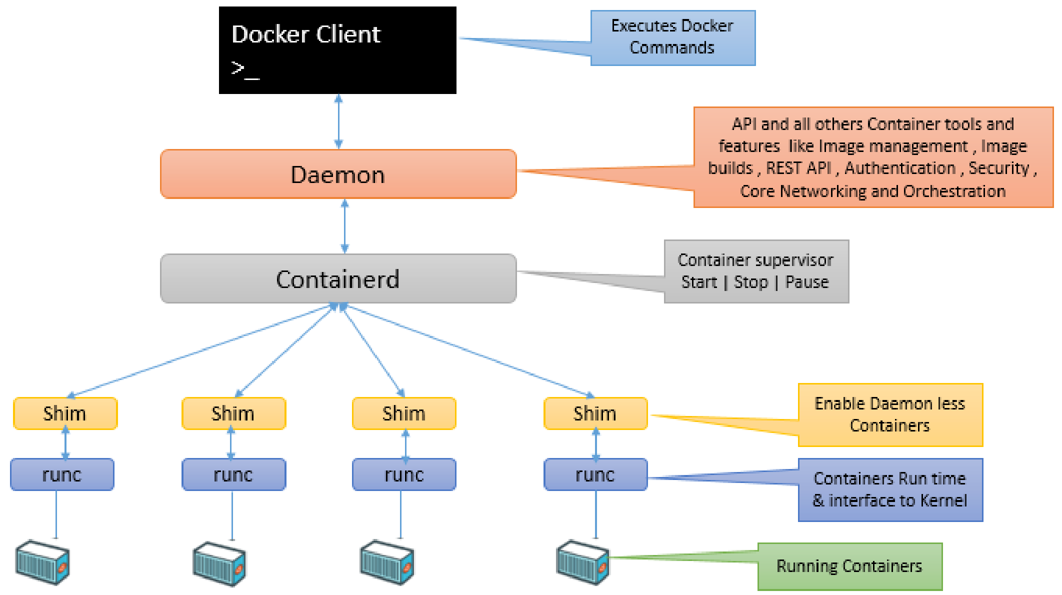
```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- Installing on Mac
  - Download and install the official Docker.dmg from docker.com
- Installing on Windows
  - WSL2 must be enabled on Windows.
  - Download the latest Docker Desktop installer from docker.com

1.4 Docker Machine

- Though Docker runs natively on Linux, it may be desirable to have two different host environment, such as Ubuntu and CentOS
- To achieve this, VMs running Docker may be used
- To simplify management of different Docker host, it is possible to use Docker Machine
- Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands
- Docker Machine enables you to provision multiple remote Docker hosts on various flavors of Linux.
- Additionally, Machine allows you to run Docker on older Mac or Windows systems as well as cloud providers such as AWS, Azure and GCP
- Using the docker-machine command, you can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host

1.5 Docker Engine Architecture


Docker Client
>_
Executes Docker Commands
Daemon
API and all others Container tools and features like Image management , Image builds , REST API , Authentication , Security , Core Networking and Orchestration
Containerd
Container supervisor Start | Stop | Pause
Shim
Shim
Shim
Shim
Enable Daemon less Containers
runc
runc
runc
runc
Containers Run time & interface to Kernel
Running Containers

Source: Adapted from https://www.dclessons.com/how-docker-engine-works

1.6 runC

- runC is a lightweight container runtime.
- It was originally a low-level Docker component.
- It has since been rolled out as a standalone modular tool.
- It’s purpose is to improve portability of containers by providing a standardized interoperable container runtime that can work both as part of Docker and independently of Docker in alternative container systems.
- runC can help you avoid being strongly tied to specific technologies, hardware or cloud service providers.

1.7 containerd

- Like runC, containerd is another core building block of the Docker system that has been separated off as an independent open-source project.
- containerd is a daemon that acts as an interface between the container engine and container runtimes.
- It is supported by both Linux and Windows
- It provides an abstracted layer, API, that makes it easier to manage container lifecycles, such as image transfer, container execution, snapshot functionality and certain storage operations, using simple API requests.
- It makes containers more portable since you don't have to rely on platform-specific, low-level system calls to manage container lifecycle

1.8 Docker as Platform-as-a-Service

- Docker defines an API for creating, deploying and managing containers that make up highly distributed systems spanning multiple physical machines
  - Docker-based systems can also efficiently run multiple isolated applications on a single physical machine
- On-demand provisioning of applications by Docker supports the Platform-as-a-Service (PaaS)–style deployment and scaling

1.9 Docker and Cloud

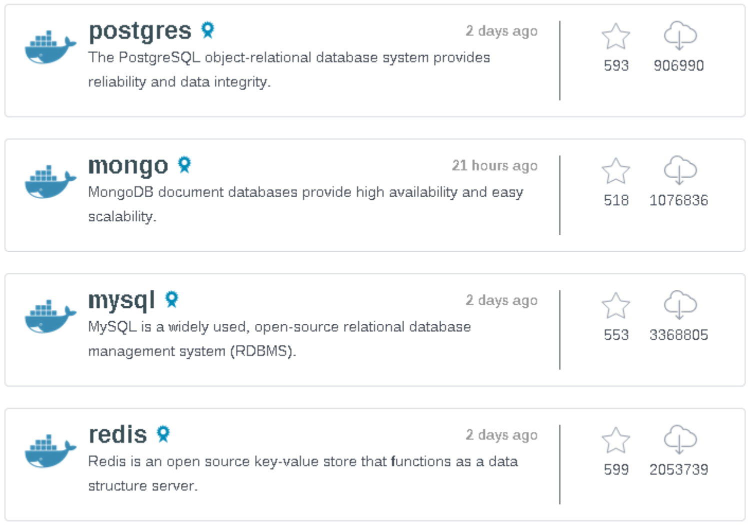
- Docker is also deployed on a number of Cloud platforms
  - Amazon Web Services (ECS/EKS)
  - Google Cloud Platform (GKE)
  - Microsoft Azure (ACS/AKS)
  - OpenStack
  - Rackspace

1.10 Docker Services

- Docker deployment model is application-centric and in this context provides the following services and tools:
  - A uniform format for bundling an application along with its dependencies which is portable across different machines
  - Tools for automatic assembling a container from source code: make, maven, Debian packages, RPMs, etc.
  - Container versioning with deltas between versions

1.11 Docker Application Container Public Repository

- Docker community maintains the repository for official and public domain Docker application images: https://hub.docker.com



postgres — 2 days ago
The PostgreSQL object-relational database system provides reliability and data integrity.
593 906990

mongo — 21 hours ago
MongoDB document databases provide high availability and easy scalability.
518 1076836

mysql — 2 days ago
MySQL is a widely used, open-source relational database management system (RDBMS).
553 3368805

redis — 2 days ago
Redis is an open source key-value store that functions as a data structure server.
599 2053739

- Note: Only use “official” images to avoid security problems. Also, check the repository location with your security team as you might not be able to use Docker Hub.

1.12 Competing Systems

- Here are some of the alternatives:
  - **LXC -** The technology was a forerunner to Docker and is sponsored by Canonical (the company behind Ubuntu)
  - **Windows Containers -** Windows Server 2016+ offers a lightweight alternative to full-blown Windows virtual machines (VMs)—Windows Containers, which take a similar abstraction approach to Docker, and Hyper-V Containers.
  - **Podman -** It is also a very close open-source alternative to Docker and uses commands that are identical to those supported by the Docker CLI, except you use podman in place of the docker base command.

1.13 Docker Command Line

- The following commands are shown as executed by the root (privileged) user:

```
docker run ubuntu:{tag} echo 'Yo Docker!'
```

  - This command will create a docker container based on the *ubuntu* image, execute the *echo* command on it, and then shuts down.

```
docker ps -a
```

  - This command will list all the containers created by Docker along with their IDs

1.14 Starting, Inspecting, and Stopping Docker Containers

`docker start -i <container_id>`

- This command will start an existing stopped container in interactive (**-i**) mode (you will get container's STDIN channel)

`docker inspect <container_id>`

- This command will provide JSON-encoded information about the running container identified by *container_id*

`docker stop <container_id>`

- This command will stop the running container identified by *container_id*

- For the Docker command-line reference, visit *https://docs.docker.com/engine/reference/commandline/cli/*

1.15 Docker Volume

- If you destroy a container and recreate it, you will lose data
- Ideally, data should not be stored in containers
- Volumes are mounted file systems available to containers
- Docker volumes are a good way of safely storing data outside a container
- Docker volumes can be shared across multiple containers
- Creating a Docker volume

```
docker volume create my-volume
```

- Mounting a volume

```
docker run -v my-volume:/my-mount-path -it ubuntu:12.04 /bin/bash
```

- Viewing all volumes

```
docker volume ls
```

- Check Docker volume location
- `docker inspect {volume_name}`
  - Typically, */var/lib/docker/volumes/{volume_name}*
- Deleting a volume

```
docker volume rm my-volume
```

1.16 Dockerfile

- Rather than manually creating containers and saving them as custom images, it's better to use Dockerfile to build images
- Sample script

```
# let's use ubuntu docker image
FROM openjdk:8

RUN apt-get update -y
RUN apt-get install sqlite -y

# deploy the jar file to the container
COPY SimpleGreeting-1.0-SNAPSHOT.jar /root/SimpleGreeting-1.0-
SNAPSHOT.jar
```

- The Dockerfile filename is case sensitive. The 'D' in Dockerfile has to be uppercase.
- Building an image using docker build. (Mind the space and period at the end of the docker build command)

```
docker build -t my-image:v1.0 .
```

- Or, if you want to use a different file name:

```
docker build -t my-image:v1.0 -f mydockerfile.txt
```

1.17 Docker Compose

- A container runs a single application. However, most modern application rely on multiple service, such as database, monitoring, logging, messages queues, etc.
- Managing a large group of containers individually is difficult
  - Especially when it comes to moving the environment from development to test to production, etc.
- Compose is a tool for defining and running multi-container Docker applications on the same host
- A single configuration file, docker-compose.yml, is used to define a group of container that must be managed as a single entity

1.18 Using Docker Compose

- Define as many Dockerfile as necessary
- Create a docker-compose.yml file that refers to the individual Dockerfile
- Sample Dockerfile

```
version: '3'
services:
  greeting:
    build: .
    ports:
    - "8080:8080"
    links:
    - mongodb
  mongodb:
    image: mongodb
    environment:
      MONGO_INITDB_ROOT_USERNAME: wasadmin
      MONGO_INITDB_ROOT_PASSWORD: secret
    volumes:
    - my-volume:/data/db
volumes:
  my-volume: {}
```

1.19 Dissecting docker-compose.yml

- The Docker Compose file should be named either docker-compose.yml or docker-compose.yaml
  - Using any other names will require to use the -f argument to specify the filename
- The docker-compose.yml file is writing in YAML
  - https://yaml.org/
- The first line, version, indicates the version of Docker Compose being used
  - As of this writing, version 3 is the latest

1.20 Specifying services

- A ‘service’ in docker-compose parlance is a container
- Services are specified under the service: node of the configuration file
- You choose the name of a service. The name of the service is meaningful within the configuration
- A service (container) can be specified in one of two ways: Dockerfile or image name
- Use build: to specify the path to a Dockerfile
- Use image: to specify the name of an image that is accessible to the host

1.21 Dependencies between containers

- Some services may need to be brought up before other services
- In docker-compose.yml, it is possible to specify which service relies on which using the links: node
- If service C requires that service A and B be brought up first, add a link as follows:

```
A:
  build: ./servicea
B:
  build: ./serviceb
C:
  build: ./servicec
  depends_on:
  - A
  - B
```

- It is possible to specify as many links as necessary
  - Circular links are not permitted (A links to B and B links to A)

1.22 Injecting Environment Variables

- In a microservice, containerized application, environment variables are often used to pass configuration to an application
- It is possible to pass environment variable to a service via the docker-compose.yml file

```
myservice:
  environment:
    MONGO_INITDB_ROOT_USERNAME: wasadmin
    MONGO_INITDB_ROOT_PASSWORD: secret
```

1.23 Summary

- Docker is a system for creating virtual environments which are, for all intents and purposes, lightweight virtual machines
- Docker containers can only run the type of OS that matches the host's OS
- Docker containers are extremely lightweight (although not so robust and secure), allowing you to achieve a higher level of deployed application density compared with traditional VMs (10x more units!)
- On-demand provisioning of applications by Docker supports the Platform-as-a-Service (PaaS)–style deployment and scaling

Chapter 2 - Introduction to Kubernetes

Objectives

Key objectives of this chapter

- What is Kubernetes?
- What Is a Container?
- Microservices and Orchestration
- Microservices and Infrastructure-as-Code
- Kubernetes Container Networking

2.1 What is Kubernetes

- Kubernetes is Greek for "helmsman" or "pilot"
- Originally founded by Joe Beda, Brendan Burns and Craig McLuckie
- Afterward, other Google engineers also joined the project
- The original codename of Kubernetes within Google was Project Seven, a reference to a Star Trek character. The seven spokes on the wheel of the Kubernetes logo is a nod to that codename
- Kubernetes is commonly referred to as **K8s**
- An open-source system for automating deployment, scaling, and management of containerized applications
- Originally designed by Google and donated to the Cloud Native Computing Foundation
- Provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts
- Supports a range of container tools, including Docker

2.2 What is a Container

- Over the past few years, containers have grown in popularity
- Containers provide process-level isolation.
- It is a computer virtualization method in which the kernel of an operating system allows the existence of multiple isolated user-space instances, instead of just one
- Such instances are called containers
- A container is a software bucket comprising everything necessary to run the software independently.
- There can be multiple containers in a single machine and containers are completely isolated from one another as well as from the host machine.
- Containers are also called virtualization engines (VEs) or Jails (e.g. FreeBSD jail)
- Containers look like real computers from the point of view of programs running in them
- Items usually bundled into a container include:
  - Application, dependencies, libraries, binaries, and configuration files

2.3 Container – Uses

- A container is useful for packaging, shipping, and deployment of any software applications that are presented as lightweight, portable, and self-sufficient containers, that will run virtually anywhere.
- It is useful for securely allocating finite hardware amongst a large number of mutually-distributing users
- System administrators may also use it for consolidating server hardware by moving services on separate hosts into containers on a single server
- Container is useful for packaging everything the app needs into a container and migrating that from one VM to another, to server or cloud without having to refactor the app.
- Container usually imposes little to no overhead, because programs in virtual partitions use the OS' normal system call interface and do not need to be subjected to emulation or be run in an intermediate virtual machines
- Container doesn't require support in hardware to perform efficiently

2.4 Container – Pros

- Containers are fast compared to hardware-level virtualization, since there is no need to boot up a full virtual machine. A Container allows you to start apps in a virtual, software-defined environment much more quickly
- The average container size is within the range of tens of MB while VMs can take up several gigabytes. Therefore a server can host significantly more containers than virtual machines
- Running containers is less resource intensive than running VMs so you can add more computing workload onto the same servers
- Provisioning containers only take a few seconds or less, therefore, the data center can react quickly to a spike in user activity.
- Containers can enable you to easily allocate resources to processes and to run your application in various environments.
- Using containers can decrease the time needed for development, testing, and deployment of applications and services.
- Testing and bug tracking also become less complicated since you there is no difference between running your application locally, on a test server, or in production.
- Containers are a very cost effective solution. They can potentially help you to decrease your operating cost (less servers, less staff) and your development cost (develop for one

consistent runtime environment).

- Using containers, developers are able to have truly portable deployments. This helps in making Continuous Integration / Continuous Deployment easier.
- Container-based virtualization are a great option for microservices, DevOps, and continuous deployment.

2.5 Container – Cons

- Compared to traditional virtual machines, containers are less secure.
  - Containers share the kernel, other components of the host operating system, and by default, they have root access.
  - This means that containers are less isolated from each other than virtual machines, and if there is a vulnerability in the kernel it can jeopardize the security of the other containers as well.
- A container offers less flexibility in operating systems. You need to start a new server to be able to run containers with different operating systems.
- Networking can be challenging with containers. Deploying containers in a sufficiently isolated way while maintaining an adequate network connection can be tricky.
- Developing and testing for containers requires training. Whereas writing applications for VMs, which are in effect the same as physical machines, is a straightforward transition for development teams.
- Single VMs often run multiple applications. Whereas containers promotes a one-container one-application infrastructure. This means containerization tends to lead to a higher volume of discreet units to be monitored and managed.

2.6 Composition of a Container

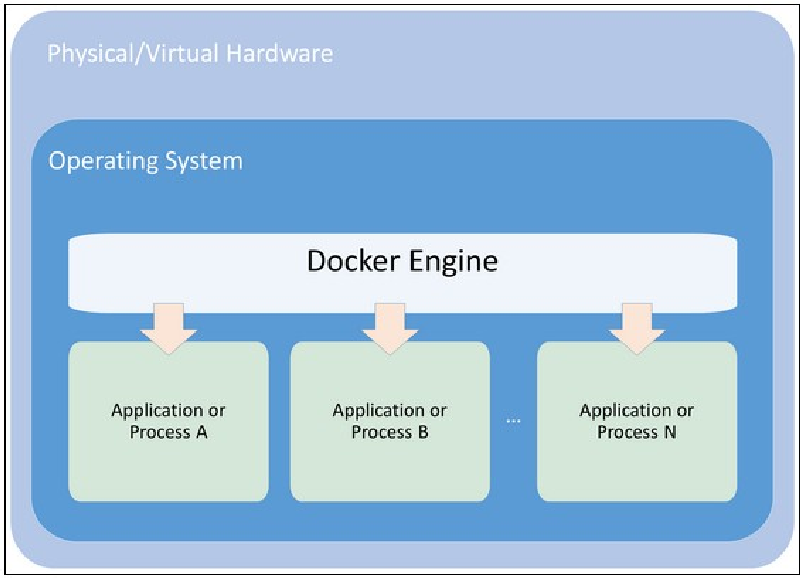
- At the core of container technology are
  - Control Groups (cgroups)
  - Namespaces
  - Union filesystems

2.7 Control Groups

- Control groups (cgroups) work by allowing the host to share and also limit the resources each process or container can consume
- This is important for both, resource utilization and security
- It prevents denial-of-service attacks on host's hardware resources

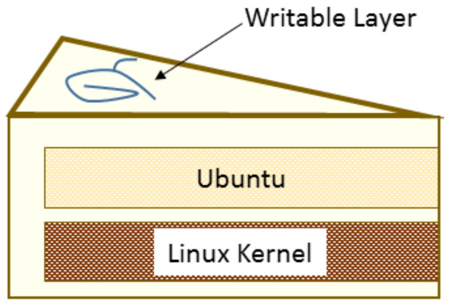
2.8 Namespaces

- Namespaces offer another form of isolation for process interaction within operating systems.
- It limits the visibility a process has on other processes, networking, filesystems, and user ID components
- Container processes are limited to see only what is in the same namespace
- Processes from containers or the host processes are not directly accessible from within the container process.

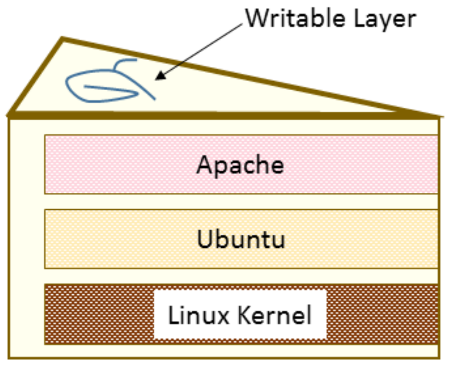


2.9 Union Filesystems

- Containers run from an image, much like an image in the VM or Cloud world, it represents state at a particular point in time.
- Container images snapshot the filesystems
- The snapshot tend to be much smaller than a VM
- The container shares the host kernel and generally runs a much smaller set of processes
- The filesystem is often layered or multi-leveled.
  - e.g. Base layer can be Ubuntu with an application such as Apache or MySQL stacked on top of it



Base Ubuntu Image



Apache Ubuntu Image

2.10 Microservices

- The microservice architectural style is an approach to developing a single application as a suite of small services
- Each service runs in its own process and communicates with lightweight mechanisms, often an HTTP resource API
- These services are built around business capabilities and independently deployable by fully automated deployment machinery

2.11 Microservices and Containers / Clusters

- Containers are excellent for microservices, as it isolates the services.
- Containerization of single services makes it easier to manage and update these services
- Docker has led to the emergence of frameworks for managing complex scenarios, such as:
  - how to manage single services in a cluster
  - how to manage multiple instances in a service across hosts
  - how to coordinate between multiple services on a deployment and management level
- Kubernetes allows easy deployment and management of multiple Docker containers of the same type through an intelligent tagging system
- With Kubernetes, you describe the characteristics of the image, e.g. number of instances, CPU, RAM, you would like to deploy

2.12 Microservices and Orchestration

- Microservices can benefit from deployment to containers
- Issue with containers is, they are isolated. Microservices might require communication with each other
- Container orchestration can be used to handle this issue
- Container orchestration refers to the automated arrangement, coordination, and management of software containers
- Container orchestration also helps in tackling challenges, such as
    - service discovery
    - load balancing
    - secrets/configuration/storage management
    - health checks, auto-[scaling/restart/healing] of containers and nodes
    - zero-downtime deploys

2.13 Microservices and Infrastructure-as-Code

- In the old days, you would write a service and allow the Operations (Ops) team to deploy it to various servers for testing, and eventually production.
- Infrastructure-as-Code solutions helps in shortening the development cycles by automating the set up of infrastructure
- Popular infrastructure-as-code solutions include Puppet, Chef, Ansible, Terraform, and Serverless.
- In the old days, servers were treated as part of the family.
  - Servers were named, constantly monitored, and carefully updated
- Due to containers and infrastructure-as-code solutions, these days the servers (containers) are often not updated. Instead, they are destroyed, then recreated.
- Containers and infrastructure-as-code solutions treat infrastructure as disposable.

2.14 Kubernetes Container Networking

- Microservices require a reliable way to find and communicate with each other.
- Microservices in containers and clusters can make things more complex as we now have multiple networking namespaces to bear in mind.
- Communication and discovery requires traversing of container IP space and host networking.
- Kubernetes benefits from getting its ancestry from the clustering tools used by Google for the past decade. Many of the lessons learned from running and networking two billion containers per week have been distilled into Kubernetes

2.15 Kubernetes Networking Challenges

- When planning Kubernetes networking, there are 4 networking problems to address:
  - Container-to-container communications: this is solved by Pods and localhost communications.
  - Pod-to-Pod communications
  - Pod-to-Service communications
  - External-to-Service communications

2.16 Kubernetes Networking Requirements

- Kubernetes enforces the following requirements on any networking implementation, except any intentional network segmentation policies
  - pods on a node can communicate with all pods on all nodes without NAT
  - agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node
  - pods in the host network of a node can communicate with all pods on all nodes without NAT

2.17 Kubernetes Networking Options

- Azure CNI for Kubernetes
  - It is an open-source plugin that integrates Kubernetes Pods with an Azure Virtual Network (aka. VNet) offering networking performance at par with VMs.
- AWS VPC CNI for Kubernetes
  - The AWS VPC CNI offers integrated AWS Virtual Private Cloud (VPC) networking for Kubernetes clusters.
  - This CNI plugin offers high throughput and availability, low latency, and minimal network lag.
- Google Compute Engine (GCE)
  - For the Google Compute Engine cluster configuration, routing is used to assign each VM a subnet. Any traffic bound for that subnet will be routed to the VM by the GCE network fabric.

2.18 Kubernetes Networking Options (contd.)

- ACI
  - Cisco Application Centric Infrastructure offers an integrated overlay and underlay SDN solution that supports containers, virtual machines, and bare metal servers.
  - ACI provides container networking integration for ACI.
- Weave
  - Provides an overlay network for Docker containers
- Flannel
  - Gives a full subnet to each host/node enabling a similar pattern to the Kubernetes practice of a routable IP per pod or group of containers
- Project Calico
  - Uses built-in routing functions of the Linux kernel.
  - It can be used for anything from small-scale deploys to large Internet-scale installations.
  - There is no need for additional NAT, tunneling, or overlays.
- Canal
  - It merges both Calico for network policy and Flannel for overlay into one solution

2.19 Kubernetes Networking – Balanced Design

- Using unique IP address at the host level is problematic as the number of containers grow.
  - Assigning an IP address to each container can also be overkill.
- In cases of sizable scale, overlay networks and NATs are needed in order to address each container.
  - Overlay networks add latency
- You have to pick between
  - fewer containers with multiple applications per container (unique IP address for each container)
  - multiple containers with fewer applications per container (Overlay networks / NAT)

2.20 Summary

- Kubernetes provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts
- Kubernetes supports a range of container tools, including Docker
- Containers are useful for packaging, shipping, and deployment of any software applications that are presented as lightweight, portable, and self-sufficient containers, that will run virtually anywhere.
- Microservices can benefit from containers and clustering.
- Kubernetes offers container orchestration

Chapter 3 - Kubernetes – From the Firehose

Objectives

Key objectives of this chapter

- Masters
- Nodes
- Pods
- Namespaces
- Resource Quota
- Authentication and Authorization
- Routing
- Registry
- Storage Volumes

3.1 What is Kubernetes?

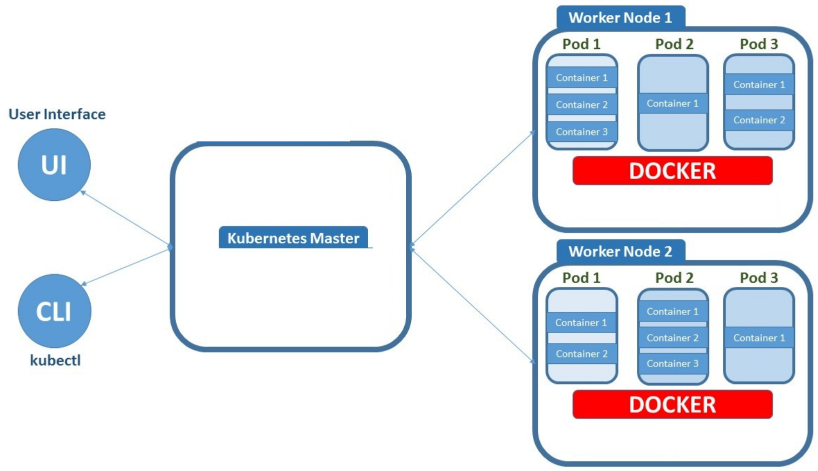
- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications
- It groups containers that make up an application into logical units for easy management and discovery.
- Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.
- Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is
- Kubernetes is open source giving you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, letting you effortlessly move workloads to where it matters to you.
- Kubernetes can be deployed on a bare-metal cluster (real machines) or on a cluster of virtual machines.

3.2 Container Orchestration

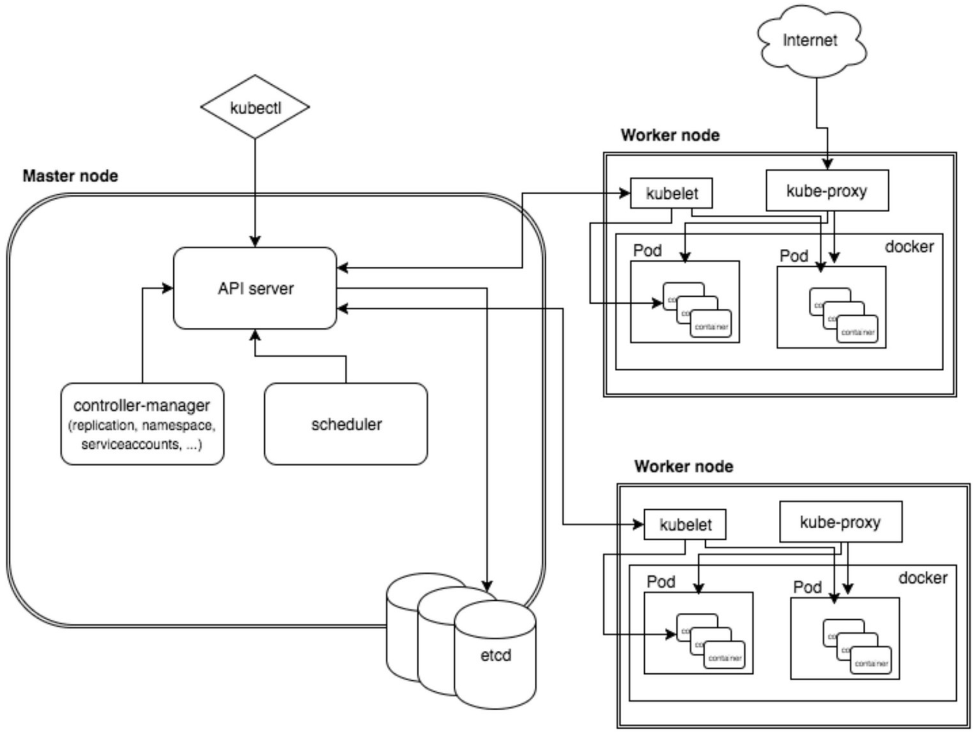
- The primary responsibility of Kubernetes is container orchestration.
- Kubernetes ensures that all the containers that execute various workloads are scheduled to run physical or virtual machines
- The containers must be packed efficiently following the constraints of the deployment environment and the cluster configuration
- Kubernetes keeps an eye on all running containers and replaces dead, unresponsive, or otherwise healthy containers.
- Kubernetes can orchestrate the containers it manages directly on bare-metal or on virtual machines
- A Kubernetes cluster can also be composed of a mix of bare-metal and virtual machines, but this is not very common.
- Containers are ideal to package microservices because, while providing isolation to the microservice, they are very lightweight compared virtual machines. This makes containers ideal for cloud deployment

3.3 Kubernetes Basic Architecture

- At a very high level, the following are the key concepts
  - Master
  - Nodes (old term used to be Minions)
  - Pods
  - Containers



3.4 Kubernetes Detailed Architecture



source: https://res.cloudinary.com/dukp6c7f7/image/upload/f_auto,fl_lossy,q_auto/s3-ghost/2016/06/o7leok.png

3.5 Kubernetes Concepts

- Cluster
- Namespace
- Master
- Node
- Pod
- Label
- Annotation

3.6 Kubernetes Concepts (contd.)

- Label Selector
- Replication Controller and replica set
- Services
- Persistent Volume
- Secret
- ConfigMap
- Workloads, such as Deployments, StatefulSet, Jobs, and DaemonSet
- Operators
- Custom Resource Definitions (CRDs)

3.7 Cluster and Namespace

- Cluster
  - A collection of physical resources, such as hosts storage and networking resources
  - The entire system may consist of multiple clusters
- Namespace
  - It is a virtual cluster
  - A single physical cluster can contain multiple virtual clusters segregated by namespaces
  - Virtual clusters can communicate through public interfaces
  - Pods can live in a namespace, but nodes can not.
  - Kubernetes can schedule pods from different namespaces to run on the same node

3.8 Node

- A single host
- It may be a physical or virtual machine
- Its job is to run pods
- Each node runs several Kubernetes components, such as a kubelet and a kube proxy
- kubelet is a service which reads container manifests as YAML files that describes a pod.
- Nodes are managed by a Kubernetes master
- The nodes are worker bees of Kubernetes and shoulder all the heavy lifting
- In the past they were called minions.

3.9 Master

- The master is the control plane of Kubernetes
- It consists of components, such as
  - API server
  - a scheduler
  - a controller manager
- The master is responsible for the global, cluster-level scheduling of pods and handling events.
- Often, all the master components are set up on a single host
- For implementing high-availability scenarios or very large clusters, you will want to have master redundancy.

3.10 Pod

- A pod is the unit of work on Kubernetes
- Each pod contains one or more containers
- Pods provide a solution for managing groups of closely related containers that depend on each other and need to cooperate on the same host
- Pods are considered throwaway entities that can be discarded and replaced at will (i.e. they are cattle, not pets)
- Each pod gets a unique ID (UID)
- All the containers in a pod have the same IP address and port space
- The containers within a pod can communicate using localhost or standard inter-process communication
- The containers within a pod have access to shared local storage on the node hosting the pod and is mounted on each container
- The benefit of grouping related containers within a pod, as opposed to having one container with multiple applications, are:
  - Transparency – making the containers within the pod visible to the infrastructure enables the infrastructure to provide services to those containers, such as process management and resource monitoring

- Decoupling software dependencies – the individual containers maybe be versioned, rebuilt, and redeployed independently
- Ease of use – users don't need to run their own process managers
- Efficiency – because the infrastructure takes on more responsibility, containers can be more lightweight

3.11 Label

- Labels are key-value pairs that are used to group together sets of objects, often pods.
- Labels are important for several other concepts, such as replication controller, replica sets, and services that need to identify the members of the group
- Each pod can have multiple labels, and each label may be assigned to different pods.
- Each label on a pod must have a unique key
- The label key must adhere to a strict syntax
  - Label has two parts: prefix and name
  - Prefix is optional. If it exists then it is separated from the name by a forward slash (/) and it must be a valid DNS sub-domain. The prefix must be 253 characters long at most
  - Name is mandatory and must be 63 characters long at most. Name must begin with an alphanumeric character and contain only alphanumeric characters, dots, dashes, and underscores. You can create another object with the same name as the deleted object, but the UIDs must be unique across the lifetime of the cluster. UIDs are generated by Kubernetes
  - Values follow the same restrictions as names

3.12 Annotation

- Unlike labels, annotation can be used to associate arbitrary metadata with Kubernetes objects.
- Kubernetes stores the annotations and makes their metadata available
- Unlike labels, annotations don't have strict restrictions about allowed characters and size limits

3.13 Label Selector

- They are used to select objects based on their labels
- A value can be assigned to a key name using equality-based selectors, (=, ==, !=).
  - e.g.
    - role = webserver
    - role = dbserver, application != sales
- **in** operator can be used as a set-based selector
  - .e.g
    - role in (dbserver, backend, webserver)

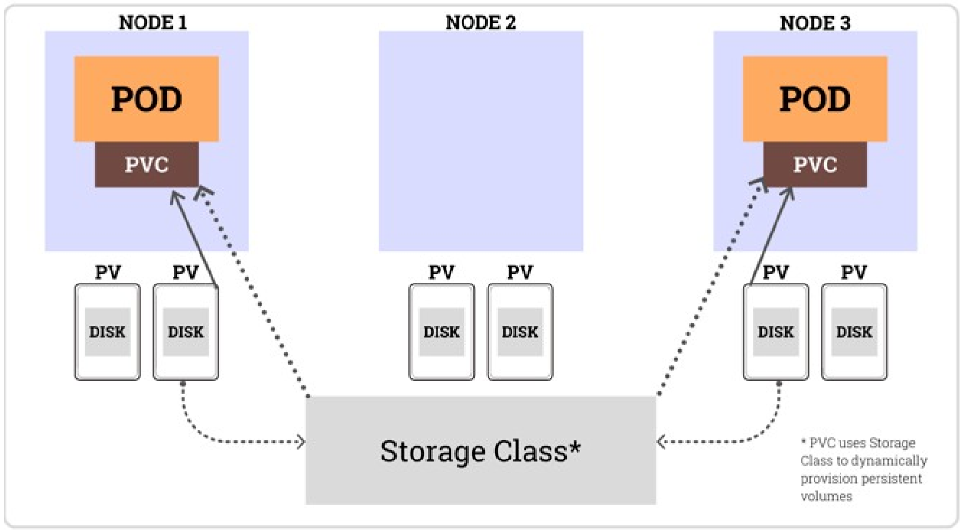
3.14 Replication Controller and Replica Set

- They both manage a group of pods identified by a label selector
- They ensure that a certain number of pods are always up and running
- Whenever the number drops due to a problem with the hosting node or the pod itself, Kubernetes fires up new instances
- If you manually start pods and exceed the specified number, the replication controller kills some extra pods
- Replication controllers test for membership by name equality, whereas replica sets can use set-based selection
- Replica sets are newer and considered as next-gen replication controllers

3.15 Service

- Services are used to expose some functionality to users or other services
- They usually involve a group of pods, usually identified by a label
- Kubernetes services are exposed through endpoints (TCP/UDP)
- Services are published or discovered via DNS, or environment variables
- Services can be load-balanced by Kubernetes

3.16 Persistent Volumes & Storage Class



source: https://blog.mayadata.io/kubernetes-storage-basics-pv-pvc-and-storageclass

3.17 Persistent Volumes & Storage Class

- When a pod is destroyed, the data used by the pod is also destroyed.
- If you want the data to outlive the pod or share data between pods, volume concept can be utilized.
- Kubernetes allows access to physical storage devices, such as SSDs, NVMe disks, NAS, and NFS servers via **Persistent Volumes** (**PV**).
- A Kubernetes Pod can consume a Persistent Volume by using a **PersistentVolumeClaim** (**PVC**) object.
- To create the PV/PVC pair for your Pod, you use a **StorageClass** object.
- With a StorageClass, you do not need to create a persistent volume separately before claiming it.

3.18 Secret

- Secrets are small objects that contain sensitive info, such as credentials
- They are stored as plain-text but can be encrypted
- They can be mounted as files into pods
- The same secret can be mounted into multiple pods
- Internally, Kubernetes creates secrets for its components, and you can create your own secrets

3.19 ConfigMaps

- A ConfigMap is an API object used to store non-confidential data in key-value pairs.
- Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
- ConfigMap does not provide encryption.
- Use a Secret rather than a ConfigMap to keep your data private.

3.20 Custom Resource Definitions (CRDs)

- Custom Resource allows you to extend Kubernetes by adding a custom resource.
- A resource is an endpoint in k8s API that allow you to store an API object of any kind.
- A custom resource allows you to create your own API objects and define your own kind just like Pod, Deployment, ReplicaSet, etc.
- Custom Resource Definition is what you use to define a Custom Resource.
- This is a powerful way to extend Kubernetes capabilities beyond the default installation.
- CRDs are YAML-based files that can be applied by using the kubectl command.

```
kubectl apply -f {my-crd.yaml}
```

3.21 Operators

- A Kubernetes operator is a software extension to Kubernetes and provides a method of packaging, deploying, and managing a Kubernetes application.
- A Kubernetes operator is an application-specific controller that extends the functionality of the Kubernetes API to create, configure, and manage instances of complex applications on behalf of a Kubernetes user.
- In Kubernetes, controllers implement control loops that repeatedly compare the desired state of the cluster to its actual state. If the cluster's actual state doesn’t match the desired state, then the controller takes action to fix the problem.
- Operator development often starts with automating the application’s installation and self-service provisioning, and follows with more complex automation capabilities.
- There is also a Kubernetes operator software development kit (SDK) that can help you develop your own operator. The SDK provides the tools to build, test, and package operators with a choice of creating operators using Helm charts, Ansible Playbooks or Golang.

3.22 Resource Quota

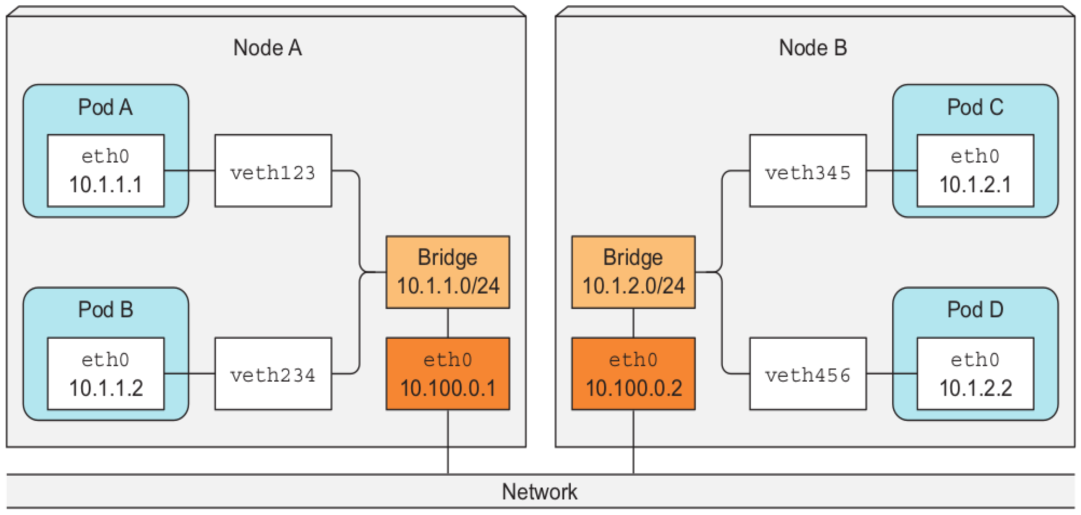
- Kubernetes allows management of different types of quota
- Compute resource quota
  - Compute resources are CPU and memory
  - You can specify a limit or request a certain amount
  - Uses fields, such as requests.cpu, requests. memory
- Storage resource quota
  - You can specify the amount of storage and the number of persistent volumes
  - Uses fields, such as requests.storage, persistentvolumeclaims
- Object count quota
  - You can control API objects, such as replication controllers, pods, services, and secrets
  - You can not limit API objects, such as replica sets and namespaces.

3.23 Authentication and Authorization

- Permission rules can be added to the Kubernetes system for more advanced management
- Applying authentication and authorization is a secure solution to prevent your data being accessed by others.
- Authentication is currently supported in the form of tokens, passwords, and certificates.
- Authorization supports three modes:
  - RBAC (Role-Based Access Control)
  - ABAC (Attribute-Based Access Control) – lets a user define privileges via attributes in a file
  - Webhook – allows for integration with third-party authorization via REST web service calls.

3.24 Routing

- Routing connects separate networks
- Routing is based on routing tables
- Routing table instructs network devices how to forward packets to their destination
- Routing is done through various network devices, such as routers, bridges, gateways, switches, and firewalls



source: https://i.stack.imgur.com/Cwd7c.png

3.25 Registry

- Container images aren't very useful if it's only available on a single machine
- Kubernetes relies on the fact that images described in a Pod manifest are available across every machine in the cluster
- Container images can be stored in a remote registry so every machine in the cluster can utilize the images.
- Registry can be public or private.
- Public registries allow anyone to download images (e.g. Docker Hub), while private registries require authentication to download images (e.g Docker Registry)
- Docker Registry is a stateless, highly scalable-server side application that stores and lets you distribute Docker images.
- Docker Registry is open-source
- Docker Registry gives you following benefits
  - tight control where your images are being stored
  - fully own your images distribution pipeline
  - enterprises often use Jfrog's artifactory, Sonatype's Nexus, CNCF's Harbor, or Redhat's Quay repositories as image repositories.

3.26 Using Docker Registry

- Default storage location is

```
/var/lib/registry
```

- Change storage location by creating an environment variable like this

```
REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/somewhere
```

- Start your registry (:2 is the version. Check Docker Hub for latest version)

```
docker run -d -p 5000:5000 –name registry registry:2.6
```

- Pull some image from the hub

```
docker pull ubuntu
```

- Tag the image so that it points to your registry

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

- Push it

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

- Pull it back

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

- Stop your registry and remove all data

```
docker stop registry && docker rm -v registry
```

3.27 Summary

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications
- The primary responsibility of Kubernetes is container orchestration.
- At a high level, Kubernetes involves Pods, Master, and Nodes
- Kubernetes also involves labels, annotations, replication controllers, replica sets, and secrets
- Container images can be deployed to a public or private registry

Chapter 4 - Kubernetes Workload

Objectives

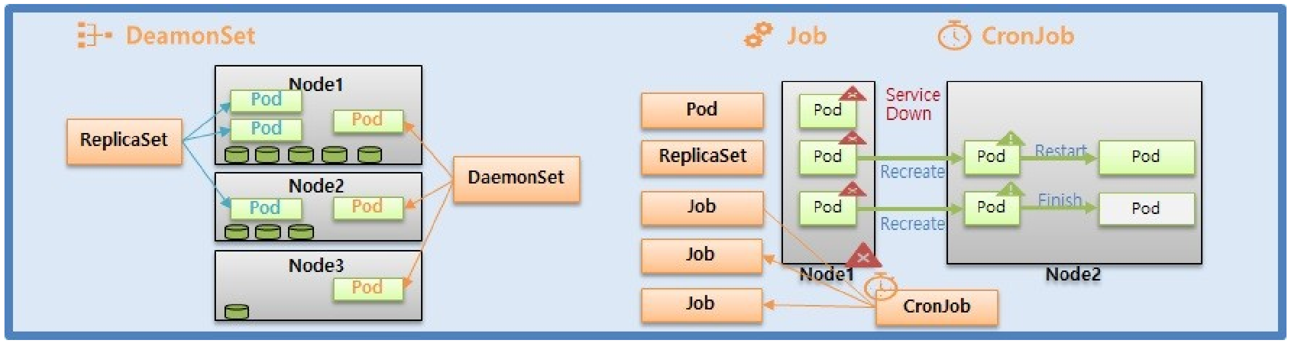
Key objectives of this chapter

- Managing Workload
- Working with Deployments
- Working with StatefulSet
- Working with Jobs
- Working with DaemonSet

4.1 Kubernetes Workload

- Workloads are objects you use to manage and run your containers on the cluster
- To deploy and manage your containerized applications and other workloads on your Kubernetes cluster, you create Kubernetes controller objects.
- The controller objects represent the applications, daemons, and batch jobs running on your clusters.
- Kubernetes provides different kinds of controller objects that correspond to different kinds of workloads you can run.
- Certain controller objects are better suited to representing specific types of workloads.
- Some common types of workloads and the Kubernetes controller objects you can create include:
  - Stateless applications
  - Stateful applications
  - Jobs
  - DaemonSets

4.2 Kubernetes Workload (contd.)



source: https://kubetm.github.io/img/practice/beginner/Controller%20with%20DatemonSet,%20Job,%20CronJob%20for%20Kubernetes.jpg

4.3 Managing Workloads

- You can create, manage, and delete objects using imperative and declarative methods.
- There are 3 ways to manage workloads:
  - ◊ imperative commands
  - ◊ imperative object configuration
  - ◊ declarative object configuration

4.4 Imperative commands

- Imperative commands allow you to quickly create, view, update and delete objects with kubectl.
- These commands are useful for one-off tasks or for making changes to active objects in a cluster.
- Imperative commands are commonly used to operate on live, deployed objects on your cluster.
- Imperative commands do not require a strong understanding of object schema and do not require configuration files.
- kubectl features several verb-driven commands for creating and editing Kubernetes objects. For example:
  - **run**: Generate a new object in the cluster. Unless otherwise specified, run creates a Deployment object. run also supports several other generators.
  - **expose**: Create a new Service object to load balance traffic across a set of labeled Pods.
  - **autoscale**: Create a new Autoscaler object to automatically horizontally scale a controller object, such as a Deployment.

4.5 Imperative Object Configuration

- Imperative object configuration creates, updates, and deletes objects using configuration files containing fully-defined object definitions.
- You can store object configuration files in source control systems and audit changes more easily than with imperative commands.
- You can run kubectl apply, delete, and replace operations with configuration files or directories containing configuration files.
- Configuration files can be in YAML or JSON format. YAML is most commonly used.
- The details are available on the official website:

  https://kubernetes.io/docs/reference/generated/kubernetes-api/

4.6 Declarative Object Configuration

- Declarative object configuration operates on locally-stored configuration files but does not require an explicit definition of the operations to be executed.
- Instead, operations are automatically detected per-object by kubectl.
- This is useful if you are working with a directory of configuration files with many different operations.
- Declarative object management requires a strong understanding of object schemas and configuration files.
- You can run kubectl apply to create and updates objects declaratively. apply updates objects by reading the whole live object, calculating the differences, then merging those differences by sending patch requests to the Kubernetes API server

4.7 Configuration File Schema

- A configuration file can involve several attributes. Here's a list of some of the commonly used attributes:
    - **apiVersion**: api version for the type of the object you are creating. It can be any of the following: v1, v1beta1, v1beta2, …. Refer to the official website to see the possible versions.
    - **kind**: the type of object you are creating, such as deployment, pod, service, job, secret, replication controller, etc.
    - **metadata**: information about the "kind". Often "name" is defined that must be unique.
    - **spec**: actual object specification is defined here.
    - **replicas**: number of replicas that will be created
    - **template:** template to be used by all the pods. e.g. label can be assigned to the pods.
    - ...

4.8 Understanding API Version

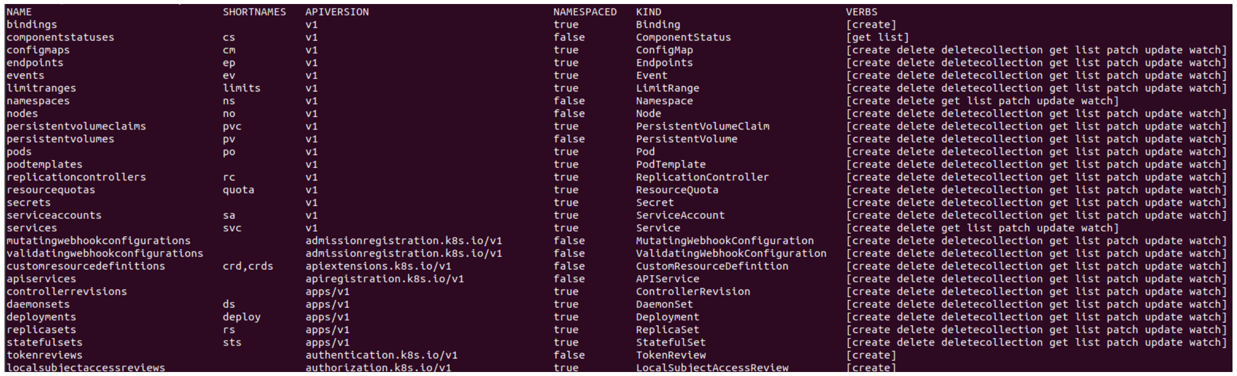
- Each Kubernetes release uses a different apiVersion.
- **alpha -** early candidate for the new release. These may contain bugs and may not be available in the final releae.
- **beta -** the beta features will eventually be included in the new release but the way objects are defined and used may change in the final relase.
- **stable -** these are safe to use and do not contain ‘alpha’ or ‘beta’ in the name.
- **v1 -** the first stable release of the Kubernetes API and contains most core objects
- **apps/v1 -** the most common API group in Kubernetes, with many core objects being drawn from it and v1. It includes objects, such as Deployments, RollingUpdates, and ReplicaSets.
- **autoscaling/v1 -** allows pods to be autoscaled based on resource utilization

4.9 Understanding API Version

- **batch/v1 -** contains objects related to batch processing and job-like tasks
- **batch/v1beta1 - a** beta release of new functionality for batch objects in Kubernetes. It also includes CronJobs that let you run Jobs at a specific time or periodicity.
- **certificates.k8s.io/v1beta1 -** validate network certificates for secure communication in your cluster
- **extensions/v1beta1 -** this group is deprecated. Before Kubernetes 1.6, it included many commonly used features of Kubernetes, such as Deployments, DaemonSets, ReplicaSets, and Ingress. These were relocated from extensions to specific API groups (e.g. apps/v1).
- **policy/v1beta1 -** adds the ability to set a pod disruption budget and new rules around pod security.
- **rbac.authorization.k8s.io/v1 -** includes extra functionality for Kubernetes role-based access control.

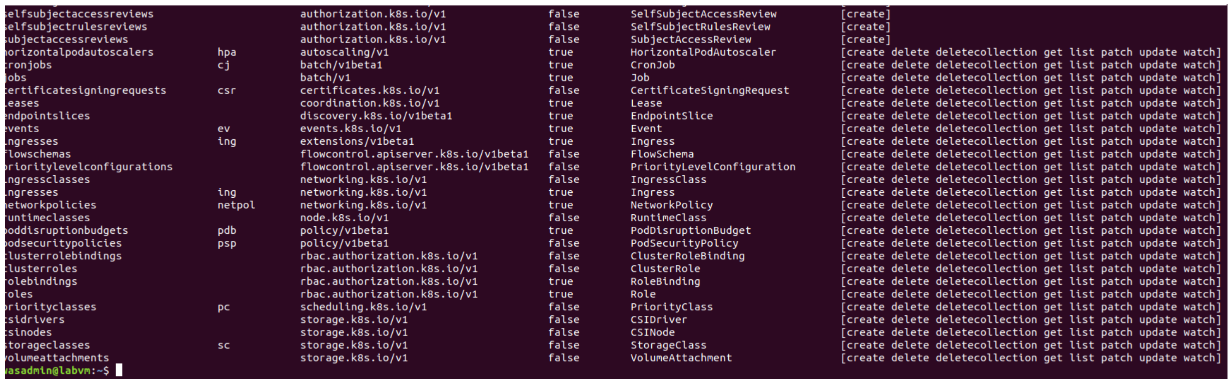
4.10 Obtaining API Versions

- Depending on your Kubernetes release, API versions would vary.
- Obtaining Kubernetes objects and API versions:
- kubectl api-resources
- Here is the sample output for Kubernetes 1.19



```
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND                             VERBS
bindings                                       v1                                     true         Binding                          [create]
componentstatuses                 cs           v1                                     false        ComponentStatus                  [get list]
configmaps                        cm           v1                                     true         ConfigMap                        [create delete deletecollection get list patch update watch]
endpoints                         ep           v1                                     true         Endpoints                        [create delete deletecollection get list patch update watch]
events                            ev           v1                                     true         Event                            [create delete deletecollection get list patch update watch]
limitranges                       limits       v1                                     true         LimitRange                       [create delete deletecollection get list patch update watch]
namespaces                        ns           v1                                     false        Namespace                        [create delete get list patch update watch]
nodes                             no           v1                                     false        Node                             [create delete deletecollection get list patch update watch]
persistentvolumeclaims            pvc          v1                                     true         PersistentVolumeClaim            [create delete deletecollection get list patch update watch]
persistentvolumes                 pv           v1                                     false        PersistentVolume                 [create delete deletecollection get list patch update watch]
pods                              po           v1                                     true         Pod                              [create delete deletecollection get list patch update watch]
podtemplates                                   v1                                     true         PodTemplate                      [create delete deletecollection get list patch update watch]
replicationcontrollers            rc           v1                                     true         ReplicationController            [create delete deletecollection get list patch update watch]
resourcequotas                    quota        v1                                     true         ResourceQuota                    [create delete deletecollection get list patch update watch]
secrets                                        v1                                     true         Secret                           [create delete deletecollection get list patch update watch]
serviceaccounts                   sa           v1                                     true         ServiceAccount                   [create delete deletecollection get list patch update watch]
services                          svc          v1                                     true         Service                          [create delete get list patch update watch]
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration     [create delete deletecollection get list patch update watch]
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration   [create delete deletecollection get list patch update watch]
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition         [create delete deletecollection get list patch update watch]
apiservices                                    apiregistration.k8s.io/v1              false        APIService                       [create delete deletecollection get list patch update watch]
controllerrevisions                            apps/v1                                true         ControllerRevision               [create delete deletecollection get list patch update watch]
daemonsets                        ds           apps/v1                                true         DaemonSet                        [create delete deletecollection get list patch update watch]
deployments                       deploy       apps/v1                                true         Deployment                       [create delete deletecollection get list patch update watch]
replicasets                       rs           apps/v1                                true         ReplicaSet                       [create delete deletecollection get list patch update watch]
statefulsets                      sts          apps/v1                                true         StatefulSet                      [create delete deletecollection get list patch update watch]
tokenreviews                                   authentication.k8s.io/v1               false        TokenReview                      [create]
localsubjectaccessreviews                      authorization.k8s.io/v1                true         LocalSubjectAccessReview         [create]
```

4.11 Obtaining API Versions (contd.)



```
selfsubjectaccessreviews                      authorization.k8s.io/v1              false        SelfSubjectAccessReview          [create]
selfsubjectrulesreviews                       authorization.k8s.io/v1              false        SelfSubjectRulesReview           [create]
subjectaccessreviews                          authorization.k8s.io/v1              false        SubjectAccessReview              [create]
orizontalpodautoscalers          hpa          autoscaling/v1                       true         HorizontalPodAutoscaler          [create delete deletecollection get list patch update watch]
ronjobs                          cj           batch/v1beta1                        true         CronJob                          [create delete deletecollection get list patch update watch]
jobs                                          batch/v1                             true         Job                              [create delete deletecollection get list patch update watch]
ertificatesigningrequests        csr          certificates.k8s.io/v1               false        CertificateSigningRequest        [create delete deletecollection get list patch update watch]
eases                                         coordination.k8s.io/v1               true         Lease                            [create delete deletecollection get list patch update watch]
ndpointslices                                 discovery.k8s.io/v1beta1             true         EndpointSlice                    [create delete deletecollection get list patch update watch]
vents                            ev           events.k8s.io/v1                     true         Event                            [create delete deletecollection get list patch update watch]
ngresses                         ing          extensions/v1beta1                   true         Ingress                          [create delete deletecollection get list patch update watch]
flowschemas                                   flowcontrol.apiserver.k8s.io/v1beta1 false        FlowSchema                       [create delete deletecollection get list patch update watch]
rioritylevelconfigurations                    flowcontrol.apiserver.k8s.io/v1beta1 false        PriorityLevelConfiguration       [create delete deletecollection get list patch update watch]
ngressclasses                                 networking.k8s.io/v1                 false        IngressClass                     [create delete deletecollection get list patch update watch]
ngresses                         ing          networking.k8s.io/v1                 true         Ingress                          [create delete deletecollection get list patch update watch]
etworkpolicies                   netpol       networking.k8s.io/v1                 true         NetworkPolicy                    [create delete deletecollection get list patch update watch]
untimeclasses                                 node.k8s.io/v1                       false        RuntimeClass                     [create delete deletecollection get list patch update watch]
oddisruptionbudgets              pdb          policy/v1beta1                       true         PodDisruptionBudget              [create delete deletecollection get list patch update watch]
odsecuritypolicies               psp          policy/v1beta1                       false        PodSecurityPolicy                [create delete deletecollection get list patch update watch]
lusterrolebindings                            rbac.authorization.k8s.io/v1         false        ClusterRoleBinding               [create delete deletecollection get list patch update watch]
lusterroles                                   rbac.authorization.k8s.io/v1         false        ClusterRole                      [create delete deletecollection get list patch update watch]
olebindings                                   rbac.authorization.k8s.io/v1         true         RoleBinding                      [create delete deletecollection get list patch update watch]
oles                                          rbac.authorization.k8s.io/v1         true         Role                             [create delete deletecollection get list patch update watch]
riorityclasses                   pc           scheduling.k8s.io/v1                 false        PriorityClass                    [create delete deletecollection get list patch update watch]
sidrivers                                     storage.k8s.io/v1                    false        CSIDriver                        [create delete deletecollection get list patch update watch]
sinodes                                       storage.k8s.io/v1                    false        CSINode                          [create delete deletecollection get list patch update watch]
storageclasses                   sc           storage.k8s.io/v1                    false        StorageClass                     [create delete deletecollection get list patch update watch]
olumeattachments                              storage.k8s.io/v1                    false        VolumeAttachment                 [create delete deletecollection get list patch update watch]
asadmin@labvm:~$
```

4.12 Stateless Applications

- A stateless application does not preserve its state and saves no data to persistent storage
- All user and session data stays with the client.
- Some examples of stateless applications include web frontends like Nginx, web servers like Apache Tomcat, and other web applications.
- You can create a Kubernetes Deployment to deploy a stateless application on your cluster.
- Pods created by Deployments are not unique and do not preserve their state, which makes scaling and updating stateless applications easier.

4.13 Sample Deployment Manifest File

```
# for versions before 1.9 use apps/v1beta2
apiVersion: apps/v1
kind: Deployment
metadata:
  # Unique key of the Deployment instance
  name: deployment-example
spec:
  # 3 Pods should exist at all times.
  replicas: 3
  template:
    metadata:
      labels:
        # Apply this label to pods and default
        # the Deployment label selector to this value
        app: nginx
    spec:
      containers:
      - name: nginx
        # Run this image
        image: nginx:1.10
       ports:
```

```
- containerPort: 80
```

4.14 Working with Deployments

- Create a deployment

```
kubectl apply -f <deployment_file>
```

- View deployment manifest

```
kubectl get deployment <deployment_name> -o yaml
```

- Get detailed information about the deployment

```
kubectl describe deployment <deployment_name>
```

- Get all pods

```
kubectl get pods
```

- Get pods that have a certain label assigned

```
kubectl get pods -l <key>=<value>
```

- Get pod details

```
kubectl describe pod <pod_name>
```

- Rolling back an update

```
kubectl rollout undo deployment <deployment_name>
```

- Scale a deployment

```
kubectl scale deployment <deployment_name> --replicas
<num_of_replicas>
```

- Delete a deployment

```
kubectl delete deployment <deployment_name>
```

4.15 Stateful Applications

- A stateful application requires that its state be saved or persistent.
- Stateful applications use persistent storage, such as persistent volumes, to save data for use by the server or by other users.
- Examples of stateful applications include databases like MongoDB and message queues like Apache ZooKeeper.
- You can create a Kubernetes StatefulSet to deploy a stateful application.
- Pods created by StatefulSets have unique identifiers and can be updated in an ordered, safe way.

4.16 Sample Stateful Manifest File

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: [STATEFULSET_NAME]
spec:
  serviceName: [SERVICE_NAME]
  replicas: 3
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app=[APP_NAME]
    spec:
      containers:
      - name: [CONTAINER_NAME]
        image: ...
        ports:
        - containerPort: 80
          name: [PORT_NAME]
        volumeMounts:
        - name: [PVC_NAME]
          mountPath: ...
  volumeClaimTemplates:
  - metadata:
      name: [PVC_NAME]
      annotations:
        ...
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

4.17 Sample Stateful Manifest File (Contd.)

- STATEFULSET_NAME] is the name you choose for the StatefulSet
- [SERVICE_NAME] is the name you choose for the Service
- [APP_NAME] is the name you choose for the application run in the Pods
- [CONTAINER_NAME] is name you choose for the containers in the Pods
- [PORT_NAME] is the name you choose for the port opened by the StatefulSet
- [PVC_NAME] is the name you choose for the PersistentVolumeClaim

4.18 Working with StatefulSet

- Create a statefulset

```
kubectl apply -f <statefulset_file>
```

- View statefulset manifest

```
kubectl get statefulset <statefulset_name> -o yaml
```

- Get detailed information about the statefulset

```
kubectl describe statefulset <statefulset_name>
```

- List PersistentVolumeClaims created

```
kubectl get pvc
```

- Get all pods

```
kubectl get pods
```

- Get pods that have a certain label assigned

```
kubectl get pods -l <key>=<value>
```

- Get pod details

```
kubectl describe pod <pod_name>
```

- Rolling back an update

```
kubectl rollout undo statefulset <statefulset_name>
```

- Scale a statefulset

```
kubectl scale statefulset <statefulset_name> --replicas
<num_of_replicas>
```

- Delete a statefulset

```
kubectl delete statefulset <statefulset_name>
```

4.19 Jobs

- Jobs represent finite, independent, and often parallel tasks which run to their completion.
- Some examples of jobs include automatic or scheduled tasks like sending emails, rendering video, and performing computations.
- You can create a Kubernetes Job to execute and manage a batch task on your cluster.
- You can specify the number of Pods that should complete their tasks before the Job is complete, as well as the maximum number of Pods that should run in parallel.

4.20 Sample Job Manifest File

- The following Job computes pi to 2000 places then prints it

```
apiVersion: batch/v1
kind: Job
metadata:
  # Unique key of the Job instance
  name: example-job
spec:
  completions: 1
  activeDeadlineSeconds: 100
  parallelism: 5
  template:
    metadata:
      name: example-job
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl"]
        args: ["-Mbignum=bpi", "-wle", "print bpi(2000)"]
      # Do not restart containers after they exit
      restartPolicy: Never
```

4.21 Sample Job Manifest File (Contd.)

- **completions** - it’s optional. By default, when a job completes, it terminates the pod. You can specify the number of times the operation should be completed before the pod is terminated.
- **activeDeadlineSeconds**: it’s optional. By default, a pod will wait indefinitely for the operation to complete.
- **parallelism**: it’s optional. By default, multiple identical jobs cannot run in parallel. You can specify the number of simultaneous jobs that can run at a given time.

4.22 Working with Batch Job

- Deploy a job

```
kubectl apply -f <job_file>
```

- View job details

```
kubectl describe job <job_name>
```

- Scale a job

```
kubectl scale job my-job --replicas=<value>
```

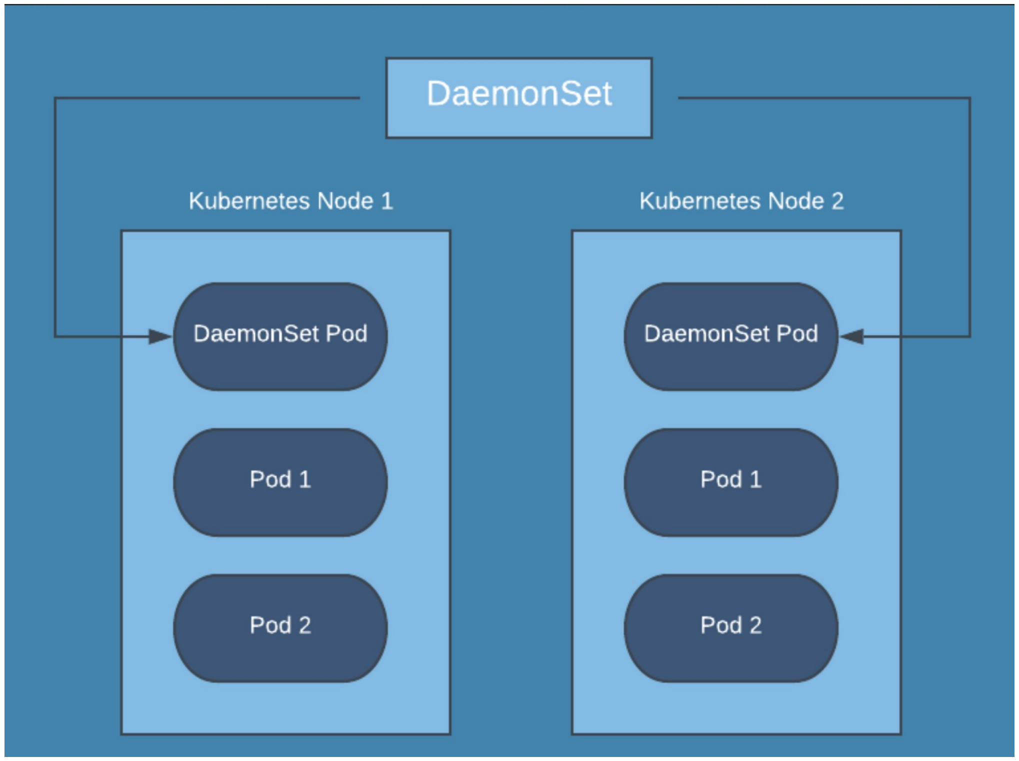
- Delete a job

```
kubectl delete job <job_name>
```

4.23 DaemonSets

- DaemonSets perform ongoing background tasks in their assigned nodes without the need for user intervention.
- Examples of DaemonSets include log collectors like Fluentd and monitoring services, such as Prometheus
- You can create a Kubernetes DaemonSet to deploy it on your cluster.
- DaemonSets create one Pod per node, and you can choose a specific node to which the DaemonSet should deploy.

4.24 DaemonSets (contd.)



source: https://www.bluematador.com/hs-fs/hubfs/blog/new/An%20Introduction%20to%20Kubernetes%20DaemonSets/DaemonSets.png?width=770&name=DaemonSets.png

4.25 Sample Daemon Manifest File

- The following configuration file is for a daemon that prints the hostname on each Node in the cluster every 10 seconds

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  # Unique key of the DaemonSet instance
  name: daemonset-example
spec:
  template:
    metadata:
      labels:
        app: daemonset-example
    spec:
      containers:
      # This container is run once on each Node in the cluster
      - name: daemonset-example
        image: ubuntu:trusty
        command:
        - /bin/sh
        args:
        - -c
        # This script is run through `sh -c <script>`
        - >-
          while [ true ]; do
          echo "DaemonSet running on $(hostname)" ;
          sleep 10 ;
          done
```

4.26 Rolling Updates

- You can perform a rolling update to update the images, configuration, labels, resource limits, and annotations of the workloads in your clusters.
- Rolling updates incrementally replace your resource's Pods with new ones, which are then scheduled on nodes with available resources.
- Rolling updates are designed to update your workloads without downtime.
- The following objects represent Kubernetes workloads.
  - DaemonSets
  - Deployments
  - StatefulSets
- You can trigger a rolling update on these workloads by updating their Pod Template.
- The Pod Template contains the following fields:
  - containers: image
  - metadata: labels
  - volumes

4.27 Rolling Updates (Contd.)

- Change object’s image

```
kubectl set image deployment nginx nginx=nginx:1.9.1
```

- Set resource limit

```
kubectl set resources deployment nginx --limits cpu=200m,memory=512Mi
--requests cpu=100m,memory=256Mi
```

- Check rollout status

```
kubectl rollout status deployment nginx
```

- Pause a rollout

```
kubectl rollout pause deployment nginx
```

- Resume rollout update

```
kubectl rollout resume deployment nginx
```

4.28 Rolling Updates (Contd.)

- View rollout history

```
kubectl rollout history deployment nginx
```

- View details of a specific revision

```
kubectl rollout history deployment nginx --revision 3
```

- Rollback an update

```
kubectl rollout undo deployments nginx
```

- Rollback to a specific revision

```
kubectl rollout undo deployment nginx --to-revision 3
```

4.29 Summary

- Some common types of workloads and the Kubernetes controller objects you can create include:
  - Stateless applications
  - Stateful applications
  - Batch jobs
  - Daemons

Chapter 5 - Scheduling and Node Management

Objectives

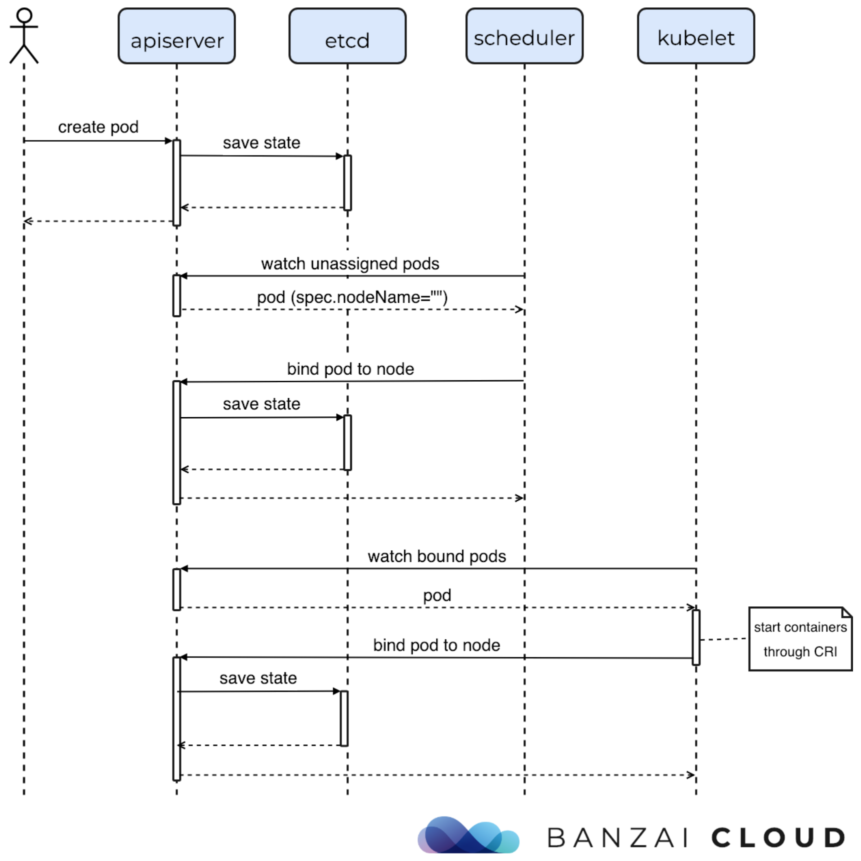
Key objectives of this chapter

- Scheduling Pods to Nodes
- Affinities

5.1 Kubernetes Scheduler

- One of the primary jobs of the Kubernetes is to schedule containers to worker nodes in the cluster of machines.
- The scheduling is accomplished by a component in the Kubernetes cluster called the Kubernetes scheduler.
- Kubernetes can handle a wide variety of workloads, such as web/API application serving, big data batch jobs, and machine learning.
- The key to ensuring that all of these very different applications can operate in on the same cluster lies in the application of job scheduling
- Kubernetes scheduler ensures that each container is placed onto the worker node best suited to it.

5.2 Kubernetes Scheduler Overview (contd.)



source: https://banzaicloud.com/blog/k8s-custom-scheduler/

5.3 Kubernetes Scheduler Overview (contd.)

- In Kubernetes, the nodeName field indicates the node on which a Pod should execute.
- When a Pod is first created, it generally doesn't have a nodeName field.
- The Kubernetes scheduler is constantly scanning for Pods that don't have a nodeName
- The Pods without the nodeName field are eligible for scheduling.
- The scheduler selects an appropriate node for the Pod and updates the Pod definition with the nodeName that the scheduler selected.
- After the nodeName is set, the kubelet running on that node is notified about the Pod's existence and it executes that Pod on that node.

5.4 Skip Kubernetes Scheduler

- To skip the scheduler, you can set the nodeName yourself on a Pod.
- This direct schedules a Pod onto a specific node.
- This is how the DaemonSet controller schedules a single Pod onto each node in the cluster.
- In general, direct scheduling should be avoided, since it tends to make your application more brittle and your cluster less efficient.
- You should trust the scheduler to make the right decision, just as you trust the operating system to find a core to execute your program when you launch it on a single machine.

5.5 Scheduling Process

- The node for a Pod is determined by several factors, some of which are supplied by the user and the others are calculated by the scheduler.
- The scheduling process relies on the following:
  - Predicates
  - Priorities

5.6 Scheduling Process - Predicates

- A predicate indicates whether a Pod fits onto a particular node.
- Predicate is a hard constraint, which, if violated, lead to a Pod not operating correctly on that node.
- For example:
  - the amount of memory requested by the Pod. If that memory is unavailable on the node, the Pod cannot get all of the memory that it needs and the constraint is violated—it is false.
  - a node-selector label query specified by the user. In this case, the user has requested that a Pod only run on certain machines as indicated by the node labels. The predicate is false if a node does not have the required label.

5.7 Scheduling Process - Priorities

- Predicates indicate situations that are either true or false—the Pod either fits or it doesn't
- Priority is an additional generic interface used by the scheduler to determine preference for one node over another.
- The role of priorities or priority functions is to score the relative value of scheduling a Pod onto a particular node.
- Unlike predicates, the priority function does not indicate whether or not the Pod being scheduled onto the node is viable
- The predicate function attempts to judge the relative value of scheduling the Pod onto that particular node.
- For example, a priority function would weight nodes where the image has already been pulled. Therefore, the container would start faster than nodes where the image is not present and would have to be pulled, delaying Pod startup.
- Ultimately, all of the various predicate values are mixed to achieve a final priority score for the node, and this score is used to determine where the Pod is scheduled.

5.8 Scheduling Algorithm

- For every Pod that needs scheduling, the scheduling algorithm is run.
- At a high level, the algorithm looks like this:

```
schedule(pod): string
    nodes := getAllHealthyNodes()
    viableNodes := []
    for node in nodes:
        for predicate in predicates:
            if predicate(node, pod):
                viableNodes.append(node)

    scoredNodes := PriorityQueue<score, Node[]>
    priorities := GetPriorityFunctions()
    for node in viableNodes:
        score = CalculateCombinedPriority(node, pod, priorities)
        scoredNodes[score].push(node)

    bestScore := scoredNodes.top().score
    selectedNodes := []
    while scoredNodes.top().score == bestScore:
      selectedNodes.append(scoredNodes.pop())

    node := selectAtRandom(selectedNodes)
    return node.Name
```

- The actual code is available on the Kubernetes GitHub page.

```
https://github.com/kubernetes/kubernetes
```

5.9 Kubernetes Scheduling Algorithm

- The scheduler gets the list of all currently known and healthy nodes.
- For each predicate, the scheduler evaluates the predicate against the node and the Pod being scheduled.
- If the node is viable, the node is added to the list of possible nodes for scheduling.
- All of the priority functions are run against the combination of Pod and node.
- The results are pushed into a priority queue ordered by score, with the best-scoring nodes at the top of the queue.
- All nodes that have the same score are popped off of the priority queue and placed into a final list.
- One of the nodes is chosen in a round-robin fashion and is then returned as the node where the Pod should be scheduled.
- Round robin is used instead of random choice to ensure an even distribution of Pods among identical nodes.

5.10 Scheduling Conflicts

- There is lag time between when a Pod is scheduled (time T1) and when the container executes (time T2)
- Due to the lag time, the scheduling decision may become invalid.
- In some cases, this may mean that a slightly less ideal node is chosen, when a better one could have been assigned.
- In general, these sorts of conflicts aren't that important and they normalize in the aggregate.
- These conflicts are thus ignored by Kubernetes.
- In case of more severe conflicts, when the node notices that it has been asked to run a Pod that no longer passes the predicates for the Pod and node, the Pod is marked as failed.
- The failed Pod doesn't count as an active member of the ReplicaSet and, thus, a new Pod will be created and scheduled onto a different node where it fits.

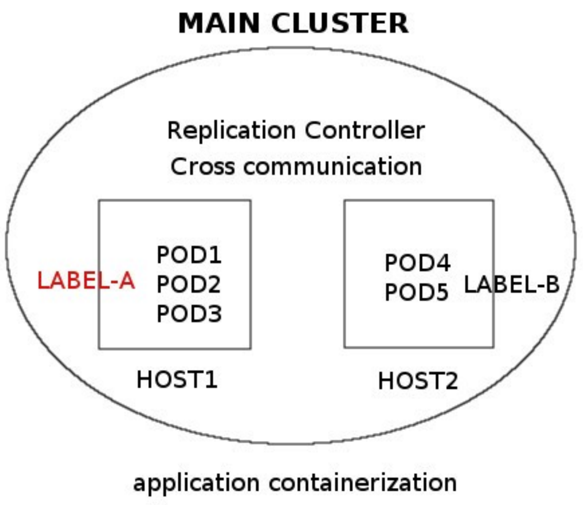
5.11 Controlling Scheduling

- Adding custom predicates and priorities is a fairly heavyweight task.
- Kubernetes provides you with several tools to customize scheduling—without having to implement anything in your code
  - Labels
  - Affinity
  - Taints
  - Tolerations

5.12 Label Selectors

- Every object in Kubernetes has an associated set of labels.
- Labels provide identifying metadata for Kubernetes objects, and label selectors are often used to dynamically identify sets of objects for various operations.
- Label selectors can also be used to identify a subset of the nodes in a Kubernetes cluster that should be used for scheduling a particular Pod.
- By default, all nodes in the cluster are potential candidates for scheduling, but by filling in the spec.nodeSelector field in a Pod or PodTemplate, the initial set of nodes can be reduced to a subset.

5.13 Label Selectors (contd.)



source: http://network-insight.net/2016/03/kubernetes-container-scheduler/

5.14 Label Selectors (Contd.)

- Add a label to a node:

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

- Add a nodeSelector field to your pod configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod-name>
spec:
  containers:
  - name: <container-name>
    image: <image-name>
  nodeSelector:
    <label-key>: <label-value>
```

- Schedule the pod on the node

```
kubectl apply -f <pod-config-yaml>
```

5.15 Node Affinity and Anti-affinity

- Node selectors provide a simple way to guarantee that a Pod lands on a particular node, but they lack flexibility.
- They cannot represent more complex logical expressions (e.g., “Label 1 is either A or B.”) nor can they represent anti-affinity (“Label 1 is A but label 2 is not C.”).
- For example, in a three-node cluster, a web application has in-memory cache such as redis. We want the web-servers to be co-located with the cache.
- Affinity is a more complicated structure to understand, but it is significantly more flexible if you want to express more complicated scheduling policies.
- You can use operators, such as In, NotIn, Exists, NotExists, Gt, and Lt

5.16 Node Affinity Example

- Consider the example just noted, in which a Pod should schedule onto a node that has label 1 with a value of either A or B. This is expressed as the following affinity policy:

```
kind: Pod
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          # label_1 == A or B
          - key: label_1
            operator: In
            values:
            - A
            - B
...
```

5.17 Node Antiaffinity Example

- To show antiaffinity, consider the policy label 1 has value A and label 2 does not equal C. This is expressed in a similar specification:

```
kind: Pod
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          # label_1 == A
          - key: label_1
            operator: In
            values:
            - A
          # label_2 != C
          - key: label_2
            operator: NotIn
            values:
            - C
...
```

5.18 Taints and Tolerations

- Node affinity is a property of pods that attracts them to a set of nodes
- Taints are the opposite as they allow a node to repel a set of pods.
- Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes.
- One or more taints are applied to a node that means the node should not accept any pods that do not tolerate the taints.
- Tolerations are applied to pods, and allow the pods to schedule onto nodes with matching taints.

5.19 Taints and Tolerations (Contd.)

- You add a taint to a node using kubectl taint.

```
kubectl taint nodes node1 key=value:<Effect>
```

- You can use the following effects:
  - **NoSchedule** - no pod will be able to schedule on to the node
  - **PreferNoSchedule** - a “preference” or “soft” version of NoSchedule – the system will try to avoid placing a pod that does not tolerate the taint on the node, but it is not required.
  - **NoExecute** - the pod will be evicted from the node (if it is already running on the node), and will not be scheduled onto the node (if it is not yet running on the node).
- To remove the taint, run the following command:

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```

- To remove all taints from a node, run the following command:

```
kubectl patch node node1.compute.internal -p '{"spec":{"taints":[]}}'
```

5.20 Taints and Tolerations (Contd.)

- Sample Toleration in a pod configuration:

```
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoSchedule"
```

5.21 Taints and Tolerations - Example

- If the node is tainted like this:

```
kubectl taint nodes node1 key1=value1:NoSchedule
kubectl taint nodes node1 key1=value1:NoExecute
kubectl taint nodes node1 key2=value2:NoSchedule
```

- … and a pod has the following tolerations:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

- … the pod will not be able to schedule onto the node, because there is no toleration matching the third taint. But it will be able to continue running if it is already running on the node when the taint is added because the third taint is the only one of the three that is not tolerated by the pod.

5.22 Summary

- Kubernetes scheduler is a component of Kubernetes that decides on what node a pod should execute.
- To gain fine-control over the scheduling, labels, affinity, taints, and tolerations can be used.

Chapter 6 - Logging, Monitoring, and Troubleshooting

Objectives

Key objectives of this chapter

- Logging in Kubernetes
- Monitoring Kubernetes
- Debugging Kubernetes
- Upgrading Kubernetes

6.1 Differences Between Logging and Monitoring

- Logging records events (e.g., a Pod being created, and monitoring records statistics (e.g., the latency of a particular request, the CPU used by a process, or the number of requests to a particular endpoint).
- Logged records, by their nature, are discrete, whereas monitoring data is a sampling of some continuous value.
- Logging systems are generally used to search for relevant information. (e.g. pod failures) For this reason, log storage systems are oriented around storing and querying vast quantities of data
- Monitoring systems are generally geared around visualization. (e.g. CPU utilization over a period of time) Thus, they are stored in systems that can efficiently store time-series data.
- Monitoring data can give you a good sense of the overall health of your cluster and can help you identify anomalous events that may be occurring.
- Logging, on the other hand, is critical for diving in and understanding what is happening, possibly across many machines, to cause such anomalous behavior.

6.2 Logging in Kubernetes

- Logs are useful for debugging problems and monitoring cluster activity.
- The easiest and most embraced logging method for containerized applications is to write to the standard output and standard error streams.
- However, the native functionality provided by a container engine or runtime is usually not enough for a complete logging solution.
- For example, if a container crashes, a pod is evicted, or a node dies, you'll usually still want to access your application's logs.
- Logs should have separate storage and lifecycle independent of nodes, pods, or containers.
- This concept is called cluster-level-logging.
- Cluster-level logging requires a separate backend to store, analyze, and query logs.
- Kubernetes provides no native storage solution for log data, but you can integrate many existing logging solutions into your Kubernetes cluster.

6.3 Basic Logging

- To read messages logged by containerized apps, use the following command:

```
kubectl logs <pod_name>
```

- To view logs of a crashed container instance, you can use the following command:

```
kubectl logs <pod_name> -previous
```

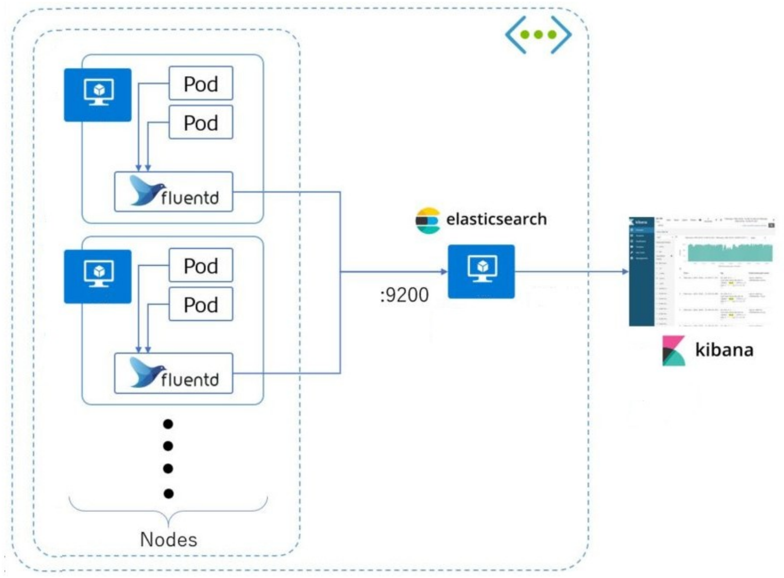
- This command fetches log entries from the container log file.
- If the container is killed and then restarted by Kubernetes, you can still access logs from the previous container.
- if the pod is evicted from the node, log files are lost.

6.4 Logging Agents

- Logging agents expose K8S logs and push them to a configured location.
- Typically, these agents are containers that have access to a directory with logs from all applications on the node.
- This solution ensures the logs are still available even if a pod or nodeis no longer available.
- Every pod in a K8S cluster has its standard output and standard error captured and stored in the /var/log/containers/ node directory.
- A logging agent is a DaemonSet type of pod that exists on each cluster node
- A logging agent captures the logs provided in each node's /var/log/containers/ directory and processes them somehow.

6.5 Fluentd and Elastic Stack

- Fluentd is a logging agent that unifies the collection and consumption of data in a pluggable manner.
- It is deployed to each node in the Kubernetes cluster, which will collect each container's log files running on that node.



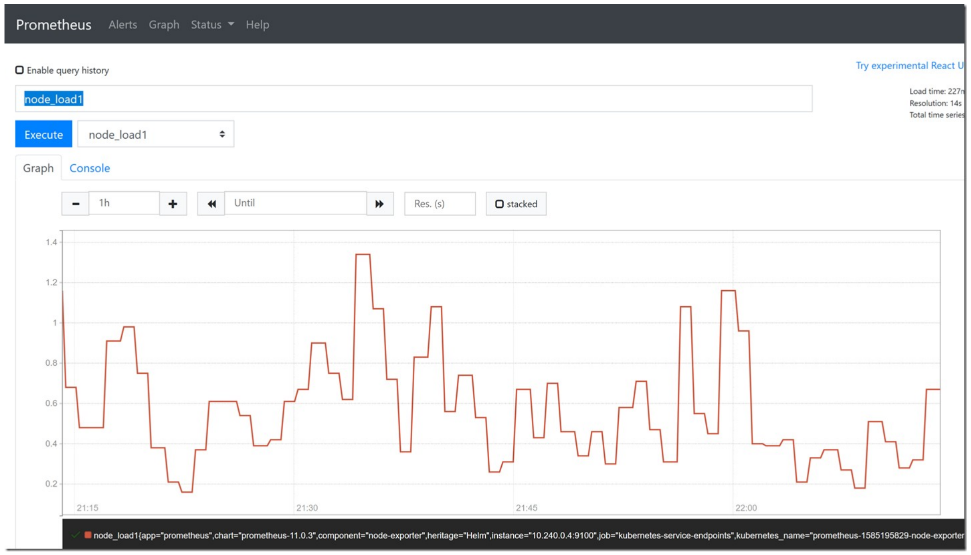
source: https://miro.medium.com/max/1794/1*eNDJDq_eWIef-XfK4l1MRg.png

- K8S acts as a log producer, Fluentd acts as a log collector, and Elastic Stack acts as a log consumer.
- To allow these components to work together, a fluentd configuration file maps how the

input data will be found and processed and where it will be saved.

6.6 Monitoring with Prometheus

- Prometheus is an open-source monitoring and alerting toolkit made for monitoring applications in clusters.
- Kubernetes features built-in support for Prometheus metrics and labels as well
- Prometheus runs as a pod in your Kubernetes cluster.
- Deploying Prometheus requires two Kubernetes objects:
  - a Deployment for the Prometheus pod itself
  - a ConfigMap that tells Prometheus how to connect to your cluster. This manifest file contains descriptions of both objects.



Load time: 227m
Resolution: 14s
Total time series

```
node_load1{app="prometheus",chart="prometheus-11.0.3",component="node-exporter",heritage="Helm",instance="10.240.0.4:9100",job="kubernetes-service-endpoints",kubernetes_name="prometheus-1585195829-node-exporter
```

6.7 Kubernetes and Prometheus - Metrics

- You can monitor various metrics, such as:
  - container_memory_usage_bytes
  - api_server_request_count
  - …
- For example:

```
container_memory_usage_bytes{image="CONTAINER:VERSION"}
```

- You can also use regular expressions when querying metrics:
- For example:

```
container_memory_usage_bytes{image=~"CONTAINER:.*"}
```

  - =~ signifies it’s a regular expression

6.8 Alerting

- Alerts can be added after monitoring is set up.
- Example of alerts includes:
  - **Whitebox:** consumption of CPU, memory, and storage
  - **Blackbox:** the latency of a request to the API server or the number of 403 (Unauthorized) responses that your API server is returning.
- Whitebox alerting offers a critical heads-up before significant problems occur.
- Blackbox alerting, on the other hand, gives you high-quality alerts caused by real, user-facing problems.

6.9 Debugging Pods

- The most basic way to check a pod status is to use the following command:

```
kubectl get pods
```

- You can retrieve a lot of information about each of the pods using:

```
kubectl describe pod <pod_name>
```

- The command shows configuration information about the container(s) and Pod (labels, resource requirements, etc.), as well as status information about the container(s) and Pod (state, readiness, restart count, events, etc.).
- The container state is one of Waiting, Running, or Terminated.
- Depending on the state, additional information will be provided

6.10 Debugging Pods (Contd.)

- If the pod state is running, the system tells you when the container started.
- Ready tells you whether the container passed its last readiness probe.
- Restart Count tells you how many times the container has been restarted
- Lastly, the command shows a log of recent events related to your Pod.
- To get more event details, use the following command:

```
kubectl get events
```

- Alternate to kubectl describe pod command is to use the following command:

```
kubectl get pod <pod_name> -o yaml
```

6.11 Debugging Nodes

- The basic way to find nodes status is to use the following command:

```
kubectl get nodes
```

- To get more details, you can use the following command:

```
kubectl describe node <node_name>
```

- Alternatively, to view the details in yaml format, use the following command:

```
kubectl get node <node_name> -o yaml
```

6.12 Debugging Replication Controllers and Services

- You can also use the following command to inspect events related to the replication controller

```
kubectl describe rc ${CONTROLLER_NAME}
```

- To troubleshoot services, ensure the service endpoints are created:

```
kubectl get endpoints <service_name>
```

6.13 Upgrading Kubernetes

- The upgrade workflow at a high-level is the following:
  - Upgrade the primary control plane node.
  - Upgrade worker nodes.
- Kubernetes Control Plane is composed of Kubernetes Master and kubelet processes.
- The Control Plane maintains a record of all of the Kubernetes Objects in the system and runs continuous control loops to manage those objects' state
- Make sure to back up any important components, such as app-level state stored in a database. Even though the upgrade process does not touch your workloads, only components internal to Kubernetes, but backups are always a best practice
- All containers are restarted after the upgrade, because the container spec hash value is changed.
- You only can upgrade from one MINOR version to the next MINOR version, or between PATCH versions of the same MINOR.
  - You cannot skip MINOR versions when you upgrade. For example, you can upgrade from 1.y to 1.y+1, but not from 1.y to 1.y+2.

6.14 Upgrade Process

- Determine which version to upgrade to
- Upgrade control plane node
  - Upgrade kubeadm
  - Get upgrade plan
  - Apply the upgrade
  - Upgrade kubelet and kubectl
  - Restart kubelet
- Upgrade worker nodes
  - Upgrade kubeadm
  - Drain the node/prepare the node for maintenance
  - Upgrade kubelet configuration
  - Upgrade kubelet and kubectl
  - Restart kubelet
  - Uncordon the node/bring the node online

6.15 Determine Which Version to Upgrade To

- Ubuntu, Debian

```
apt update
apt-cache policy kubeadm
```

- CentOS, RHEL

```
yum list --showduplicates kubeadm --disableexcludes=kubernetes
```

6.16 Upgrade kubeadm

- kubeadm should be upgraded on the control plane and all worker nodes
- Upgrade it on control plane node first followed by the worker nodes
- Ubuntu, Debian

```
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=<version> && \
apt-mark hold kubeadm
```

- CentOS, RHEL

```
yum install -y kubeadm-<version> --disableexcludes=kubernetes
```

6.17 Upgrade Control Plane Node

- Get the upgrade plan

```
sudo kubeadm upgrade plan
```

- It will display output like this: (Your versions may vary)

```
COMPONENT    CURRENT       AVAILABLE
Kubelet      1 x v1.14.2   v1.15.0
```

- Apply the upgrade

```
sudo kubeadm upgrade apply <version>
```

6.18 Upgrade kubelet and kubectl

- Ubuntu, Debian

```
apt-mark unhold kubelet kubectl && \

apt-get update && apt-get install -y kubelet=<version> kubectl=<version>
&& \

apt-mark hold kubelet kubectl
```

- CentOS, RHEL

```
yum install -y kubelet-<version> kubectl-<version> --
disableexcludes=kubernetes
```

- Restart the kubelet

```
sudo systemctl restart kubelet
```

6.19 Upgrade Worker Nodes

- The upgrade procedure on worker nodes should be executed one node at a time or few nodes at a time, without compromising the minimum required capacity for running your workloads
- Drain the node i.e. prepare the node for maintenance by marking it unschedulable and evicting the workload

```
kubectl drain $NODE --ignore-daemonsets
```

- Upgrade kubelet configuration

```
sudo kubeadm upgrade node
```

- Upgrade kubelet and kubectl on worker nodes (already covered in one of the previous slides)
- Restart the kubelet (already covered in one of the previous slides)
- Uncordon or bring the node online by marking it schedulable

```
kubectl uncordon $NODE
```

6.20 Recovering From a Failure State

- If kubeadm upgrade fails and does not roll back, for example because of an unexpected shutdown during execution, you can run the following command again:

```
kubeadm upgrade
```

- The command is idempotent and eventually makes sure that the actual state is the desired state you declare.
- To recover from a bad state, you can also run the following command:

```
kubeadm upgrade --force
```

- The command recovers a node without changing the version that your cluster is running

6.21 Summary

- Logging and monitoring are critical components of understanding how your cluster and your applications are performing
- Centralized logging and monitoring are not part of Kubernetes. You can use various logging and monitoring solutions, such as ELK (Elasticsearch, Logstash, Kibana), EFK (Elasticsearch, Fluentd, Kibana), and Splunk
- Kubernetes can be upgraded to a newer version