TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Containerization, Kubernetes

Wes J. Lloyd
School of Engineering and Technology
University of Washington – Tacoma

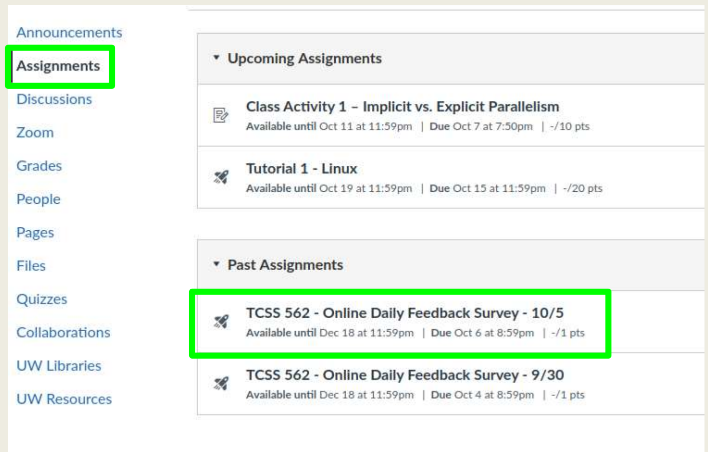
MW 5:50-7:50 PM

OBJECTIVES – 11/23

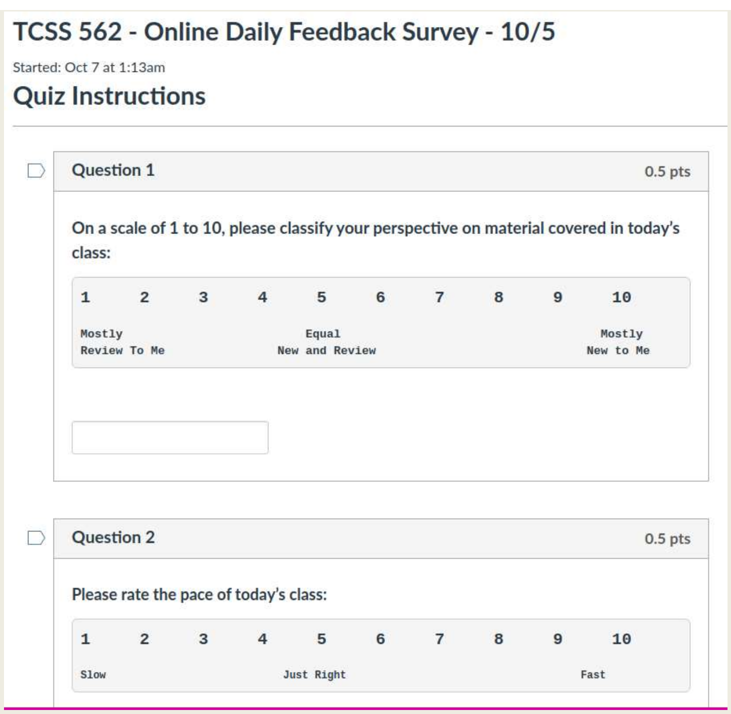
- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7
- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Take After Each Class
- Extra Credit for completing



November 23, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L15.3



November 23, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L15.4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 6.30 (↑ - *previous 6.09*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.35 (↓ - *previous 5.55*)

FEEDBACK FROM 11/18

- ***Can you please help me to understand how data travels between cloud regions?***
- ***Is it the same as how the internet works (like packets travelling over the network) or any different way?"***

- Yes, travel from one cloud region to another will typically be routed over the Internet as regions span considerable distances (thousands of KMs)
- Please note there are substantial data egress charges
- S3 to off-cloud: 9¢/GB
- S3 to distant regions: 2 ¢/GB
- S3 to local regions (Ohio→Virginia): 1 ¢/GB

FEEDBACK - 2

- ***Is 3-tier architecture still the dominant architecture in cloud?***
- Three tier architecture?
- Do you mean model-view-controller?

- ***How is that different from service oriented architecture in cloud?***
- Model-view-controller is one possible way to compose components of an service oriented application
- **Model:** data persistence
- **View:** webservices to render web-based GUI
- **Controller:** webservices to implement application logic

TUTORIAL QUESTIONS

- Tutorial 5: Thursday Nov 19th @ 11:59p

- Tutorial 6: Tuesday Nov 24th @ 11:59p

UPCOMING TUTORIALS

- Tutorial 7– [POSTED] Introduction to Docker Containerization
- **Extra credit tutorials** – *submit by Dec 18 @ 11:59p*
- Tutorial 8 – Introduction to FaaS IV: Step Functions and SQS
- Tutorial 9 – Asynchronous Function Profiling with SAAF
- **Ungraded tutorials:**
- Tutorial 10 – Automating Experiments with SAAF & FaaS Runner
- Tutorial 11 – Scaling beyond a single client – concurrent webservice benchmarking with multiple EC2 instances

November 23, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.9

OBJECTIVES – 11/23

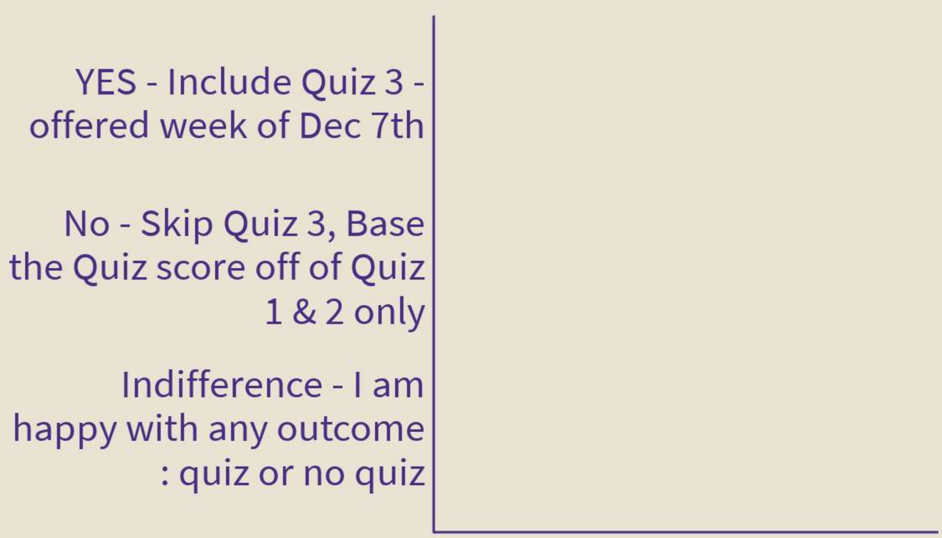
- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7
- **2^{nd} hour:**
- Introduction to Kubernetes
- Tutorial questions
- Team planning

November 23, 2020	TCSS562:Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.10

QUIZ 2

- ***Hello Professor – I am not happy with my quiz score, can I retake the quiz?***
- Quiz grade from Canvas is temporary
- Partial credit will be applied for questions having second/third best answers based on complete review of all quiz attempts the class completes the quiz
- Final quiz score will likely be curved
  - This helps to adjust for poorly written/understood questions
- Tutorials 8 & 9 will be available for extra credit

Should TCSS 562 Fall 2020 include a Quiz 3?



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

OBJECTIVES – 11/23

- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7

- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

OBJECTIVES – 11/23

- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7

- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

GROUP PRESENTATION

- TWO OPTIONS:
- ***Cloud technology presentation***
- ***Cloud research paper presentation***
  - Recent & suggested papers will be posted at: http://faculty.washington.edu/wlloyd/courses/tcss562/papers/
- Submit presentation type and topics (paper or technology) with desired dates of presentation via Canvas by: ***Monday November 23rd @ 11:59pm***
- Presentation dates:
  - Monday November 30, Wednesday December 2
  - Monday December 7, Wednesday December 9

November 23, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.15

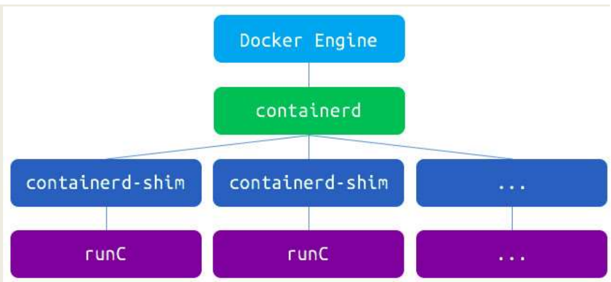
OBJECTIVES – 11/23

- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7
- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

November 23, 2020	TCSS562:Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.16

CONTAINERIZATION

OTHER DOCKER TOOLS



- **Docker Machine:** automatically provision and manage sets of docker hosts to form a cluster
- **Docker Swarm**: Clusters multiple docker hosts together to manage as a cluster.
- **Docker Compose**: Config file (YAML) for multi-container application; Describes how to deploy and configure multiple containers

CONTAINER ORCHESTRATION FRAMEWORKS

- Framework(s) to deploy multiple containers
- Provide container clusters using cloud VMs
- Similar to "private clusters"
- Reduce VM idle CPU time in public clouds
- Better leverage "sunk cost" resources
- Compact multiple apps onto shared public cloud infrastructure
- Generate to cost savings
- Reduce vendor lock-in

KEY ORCHESTRATION FEATURES

- Management of container hosts
- Launching set of containers
- Rescheduling failed containers
- Linking containers to support workflows
- Providing connectivity to clients outside the container cluster
- Firewall: control network/port accessibility
- Dynamic scaling of containers: horizontal scaling
  - Scale in/out, add/remove containers
- Load balancing over groups of containers
- Rolling upgrades of containers for application

CONTAINER ORCHESTRATION FRAMEWORKS - 2

- Docker swarm
- Apache mesos/marathon
- Kubernetes
  - Many public cloud provides moving to offer Kubernetes-as-a-service
- Amazon elastic container service (ECS)
- Apache aurora

- Container-as-a-Service
  - Serverless containers without managing clusters
  - Azure Container Instances, AWS Fargate…

OBJECTIVES – 11/23

- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7

- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

TUTORIAL #7

DOCKER, CGROUPS, RESOURCE ISOLATION

TUTORIAL COVERAGE

- Docker CLI → Docker Enginer (dockerd) → containerd → runc

- Concepts:
- Docker installation
- Working with docker files
- Docker run – create a container
- Docker ps – list containers
- Docker exec –it – run a process in an existing container
- Docker stop –stop container

Docker CLI

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  deploy      Deploy a new stack or update an existing stack
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

TUTORIAL 7

- Linux performance benchmarks
- stress-ng
- 100s of CPU, memory, disk, network stress tests
- Sysbench
- Used in tutorial for memory stress test

WE WILL RETURN AT ~6:56PM

OBJECTIVES – 11/23

- Questions from 11/18
- Quiz 2– extended until Wed 11/25 @ 11:59p
- No Office Hours 11/27
- Class on 11/25:
  Office hours, and finish any remaining lecture from today
- Introduction to Containerization cont'd
- Tutorial 7

- 2nd hour:
- Introduction to Kubernetes
- Tutorial questions
- Team planning

November 23, 2020	TCSS562:Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.28

KUBERNETES

from: "The Kubernetes Book", Nigel Poulton and Pushkar Joglekar, Version 7.0, September 2020

L15.29

KUBERNETES

- Name is from the Greek word meaning Helmsman
  - The person who steers a seafaring ship
  - The logo reinforces this theme
- Kubernetes is also sometimes called K8s
- Kubernetes is an application orchestrator
- Most common use case is to containerize cloud-native microservices applications
- What is an orchestrator?
  - System that deploys and manages applications

November 23, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.30

KUBERNETES – 2

Why does Google want to give Kubernetes away for free?

- Initially developed by Google
- **Goal:** *make it easier for potential customers to use Google Cloud*
- Kubernetes leverages knowledge gained from two internal container management systems developed at Google
  - Borg and Omega
- Google donated Kubernetes to the Cloud Native Computing Foundation in 2014 as an open-source project
- Kubernetes is written in Go (Golang)
- Kubernetes is available under the Apache 2.0 license
- Releases were previously maintained for only 8 months!
- Starting w/ v 1.19 (released Aug 2020) support is 1 year

GOALS OF KUBERNETES

1. Deploy your application
2. Scale it up and down dynamically according to demand
3. Self-heal it when things break
4. Perform zero-downtime rolling updates and rollbacks

- These features represent automatic infrastructure management
- Containerized applications run in container(s)
- Compared to VMs, containers are thought of as being:
  - Faster
  - More light-weight
  - More suited to rapidly evolving software requirements

CLOUD NATIVE APPLICATIONS

- Applications designed to meet modern software requirements including:
  - **Auto-scaling**: resources to meet demand
  - **Self-healing**: *required for high availability (HA) and fault tolerance*
  - **Rolling software updates**: with no application downtime for DevOPS
  - **Portability**: can run anywhere there's a Kubernetes cluster

WHAT IS A MICROSERVICES APP?

- Application consisting of many specialized parts that communicate and form a meaningful application
- Example components of a microservice eCommerce app:

Web front-end	Catalog service
Shopping cart	Authentication service
Logging service	Persistent data store

- **KEY IDEAS:**
- Each microservice can be coded/maintained by different team
- Each has its own release cadence
- Each is deployed/scaled separately
- Can patch & scale the log service w/o impacting others

KUBERNETES - 3

- Provides "an operating system for the cloud"
- Offers the de-facto standard platform for deploying and managing cloud-native applications
- OS: abstracts physical server, schedules processes
- Kubernetes: ***abstracts the cloud***, schedules microservices
- Kubernetes abstracts differences between private and public clouds
- Enable cloud-native applications to be cloud agnostic
  - i.e. they don't care *WHAT* cloud they run on
  - Enables fluid application migration between clouds
- Kubernetes provides rich set of tools/APIs to introspect (observe and examine) your apps

KUBERNETES - 4

- Features:
- A "control plane" – brain of the cluster
  - Implements autoscaling, rolling updates w/o downtime, self-healing
- A "bunch of nodes" – workers (muscle) of the cluster
- Provides orchestration
- The process of organizing everything into a useful application
- And also keeping it running smoothly

KUBERNETES - CLUSTER MANAGEMENT

- Master node(s) manage the cluster by:
  - Making scheduling decisions
  - Performing monitoring
  - Implementing changes
  - Responding to events
- *Masters implement the control plane of a Kubernetes cluster*
- Recipe for deploying to Kubernetes:
- Write app as independent microservices in preferred language
- Package each microservice in a container
- Create a manifest to encapsulate the definition of a Pod
- Deploy Pods to the cluster w/ a higher-level controller such as "Deployments" or "DaemonSets"

DECLARATIVE SERVICE APPROACH

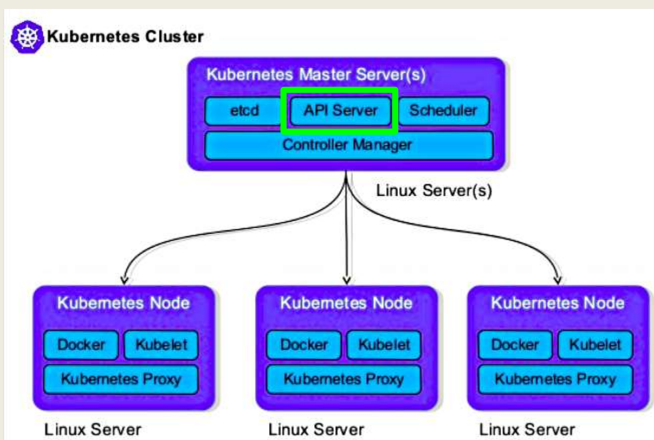
- **Imperative definition**: sets of commands and operations
  - Example: BASH script, Dockerfile
- **Declarative definition**: specification of a service's properties
  - What level of service it should sustain, etc.
  - Example: Kubernetes YAML files
- Kubernetes manages resources **declaratively**
- How apps are deployed and run are defined with YAML files
- YAML files are POSTed to Kubernetes endpoints
- Kubernetes deploys and manages applications based on declarative service requirements
- If something isn't as it should be: *Kubernetes automatically tries to fix it*

KUBERNETES MASTERS

- Provide system services to host the control plane
- Simplest clusters use only 1 master – no replication
  - Suitable for lab and dev/test environments
- Production environments: masters are replicated ~3-5x
  - Provides fault tolerance and high availability (HA)
  - Cloud-based managed Kubernetes services offer HA deployments

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



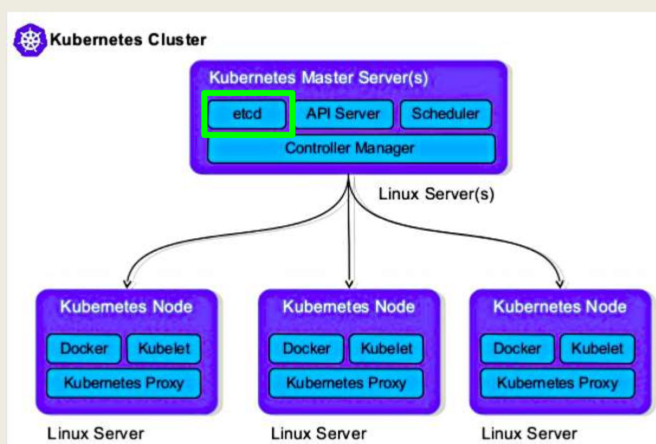
API SERVER

- Can run on 1-node for lab, test/dev environments
- Default port is 443
- Exposes a RESTful API where YAML configuration files are POST(ed) to
- YAML files (manifests) describe desired state of an application
  - Which container image(s) to use
  - Which ports to expose
  - How many POD replicas to run

November 23, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.41

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



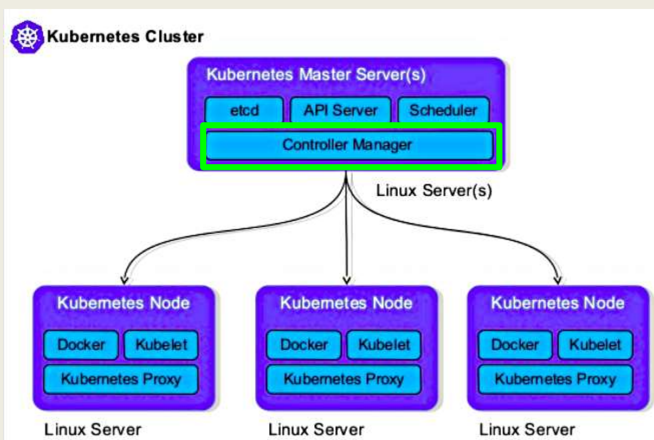
November 23, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L15.42

CLUSTER STORE

- Used to persist Kubernetes cluster state
- Persistently stores entire configuration and state of the cluster
- Currently implemented with **etcd**
  - Popular distributed key/value store (db) supporting replication
  - HA deployments may use ~3-5 replicas
  - Is the authority on true state of the cluster
- etcd prefers consistency over availability
- etcd failure: apps continue to run, nothing can be reconfigured
- Consistency of writes is vital
- Employs RAFT consensus protocol to negotiate which replica has correct view of the system in the event of replica failure

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



CONTROLLER MANAGER

- Provides a "controller" of the controllers
  - Implements background control loops to monitor cluster and respond to events
  - Control loops include: node controller, endpoints controller, replicaset controller, etc…
- **GOAL**: ***ensure cluster current state matches desired state***
- Control Loop Logic:

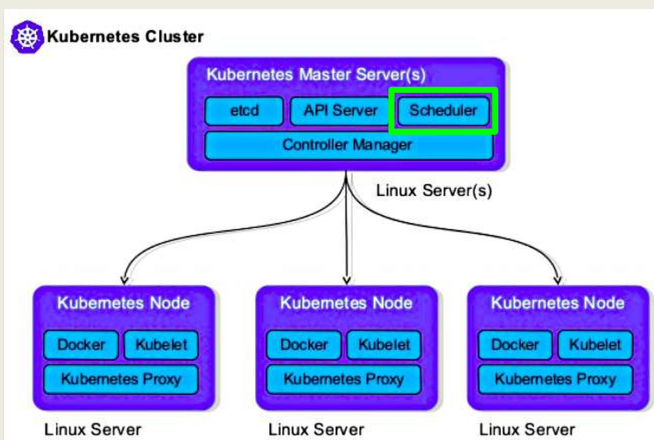
1. Obtain desired state (defined in manifest YAMLs)
2. Observe the current state
3. Determine differences
4. Reconcile differences

- Controllers are specialized to manage a specific resource type
  - They are not aware/concerned with of other parts of the system

November 23, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L15.45

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



November 23, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L15.46

TASK SCHEDULER

- Scheduler's job is to identify the best node to run a task
  - Scheduler does not actually run tasks itself
- Assigns work tasks to appropriate healthy nodes
- Implements complex logic to filter out nodes incapable of running specified task(s)
- Capable nodes are ranked
- Node with highest ranking is selected to run the task

ENFORCING SCHEDULING PREDICATES

- Scheduler performs predicate (property) checks to verify how/where to run tasks
  - Is a node tainted?
  - Does task have affinity (deploy together), anti-affinity (separation) requirements?
  - Is a required network port available on the node?
  - Does node have sufficient free resources?
- Nodes incapable of running the task are eliminated as candidate hosts

RANKING NODES

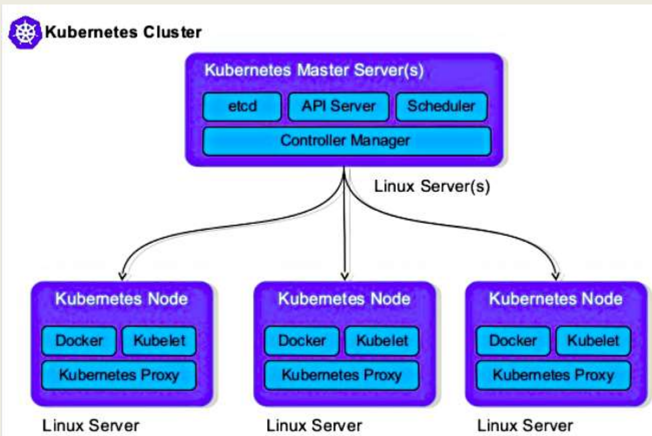
- Remaining nodes are ranked based on for example:

1. Does the node have the required images?
   - Cached images will lead to faster deployment time
2. How much free capacity (CPU, memory) does the node have?
3. How many tasks is the node already running?

- Each criterion is worth points
- ***Node with most points is selected***
- If there is no suitable node, task is not scheduled, but marked as pending
- **PROBLEM**: *There is no one-sized fits all solution to selecting the best node. How weights are assigned to conditions may not reflect what is best for the task*

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller

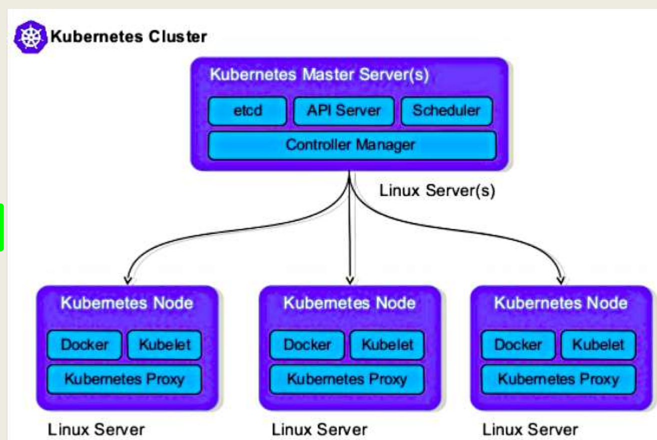


CLOUD CONTROLLER MANAGER

- Abstracts and manages integration with specific cloud(s)
- Manages vendor specific cloud infrastructure to provide instances (VMs), load balancing, storage, etc.
- Support for AWS, Azure, GCP, Digital Ocean, IBM, etc.

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



WORKER NODES

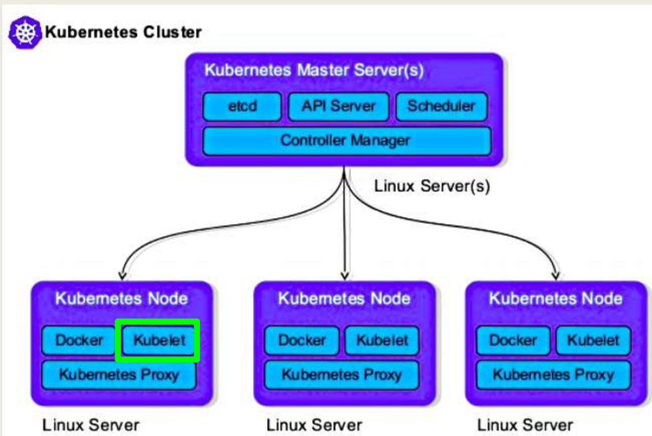
- Nodes perform tasks (i.e. host containers & services)
- Three primary functions:

1. Wait for the scheduler to assign work
2. Execute work (host containers, etc.)
3. Report back state information, etc.

- Nodes are considerably simpler than masters

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy

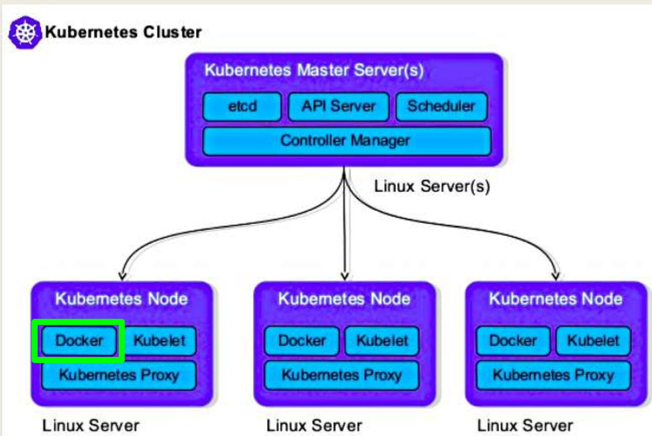


KUBELET

- Main Kubernetes agent
- Runs on every node
- Adding a new node installs the kubelet onto the node
- Kubelet registers the node with the cluster
- Monitors API server for new work assignments
- Maintains reporting back to control plane
- When a node can't run a task, kubelet is NOT responsible for finding an alternate node

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy

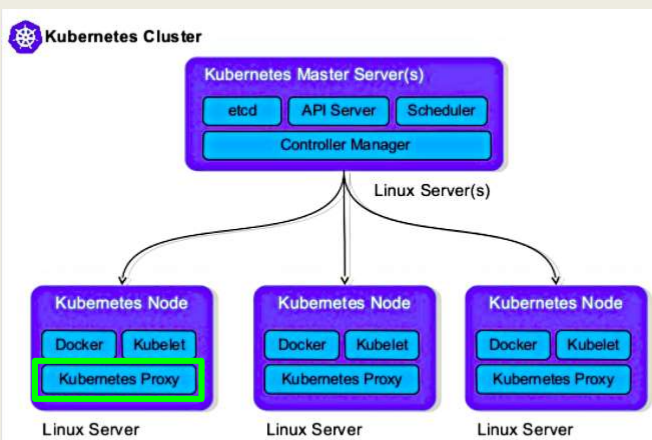


CONTAINER RUNTIME(S)

- Each node requires a container runtime to run containers
- Early versions had custom support for a limited number of container types, e.g. Docker
- Kubernetes now provides a standard Container Runtime Interface (CRI)
- CRI exposes a clean interface for 3rd party container runtimes to plug-in to
- Popular container runtimes: Docker, containerd, Kata

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy



KUBE-PROXY

- Runs on every node in the cluster
- Responsible for managing the cluster's networking
- Ensures each node obtains a unique IP address
- Implemented local IPTABLES and IPVS rules to route and load-balance traffic

- IPTABLES (ipv4) – enables configuration of IP packet filtering rules of the Linux kernel firewall

- IPVS – IP Virtual Server: provides transport-layer (layer 4) load balancing as part of the Linux kernel; Configured using ipvsadm tool in Linux

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

KUBERNETES DNS

- Every Kubernetes cluster has an internal DNS service
- Accessed with a static IP
- Hard-coded so that every container can find it
- Every service is registered with the DNS so that all components can find every Service on the cluster by NAME
- Is based on CoreDNS (https://coredns.io)

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

PODS

- Pod – atomic unit of deployment & scheduling in Kubernetes
- A Kubernetes Pod is defined to run a containerized application
- Kubernetes manages Pods, not individual containers
- Cannot run a container directly on Kubernetes
- All containers run through Pods
- Pod comes from "pod of whales"
- Docker logo shows a whale with containers stacked on top
- Whale represents the Docker engine that runs on a single host
- Pods encapsulate the definition of a single microservice for hosting purposes
- Pods can have a single container, or multiple containers if the service requires more than one

PODS - 2

- Examples of multi-container Pods:
  - Service meshes
  - Web containers with a helper container that pulls latest content
  - Containers with a tightly coupled log scraper or profiler
- YAML manifest files are used to provide a declarative description for how to run and manage a Pod
- To run a pod, POST a YAML to the API Server:
  "kubectl run <NAME>" where NAME is the service
- A Pod runs on a single node (host)
- Pods share:
  - Interprocess communication (IPC) namespace
  - Memory, Volumes, Network stack

PODS - 3

- Pods provide a "fenced" environment to run containers
- Provide a "sandbox"
- Only tightly coupled containers are deployed with a single pod
- Best practice: decouple individual containers to separate pods
  - *What is the best container composition into pods?* (1:1, 1:many)
- **Scaling**
  - Pods are the unit of scaling
  - Add and remove pods to scale up/down
  - Do not add containers to a pod, add pod instances
  - Pod instances can be scheduled on the same or different host
- **Atomic Operation**
  - Pods are either fully up and running their service (i.e. port open/exposed), or pods are down / offline

PODS - 4

- **Pod Lifecycle**
  - An application should not be tightly bound or dependent on a specific Pod instance
  - Pods are designed to fail and be replaced
  - Use of **service objects** in Kubernetes help decouple pods to offer resiliency upon failure
- **Deployments**
  - Higher level controllers often used to deploy pods
  - Controllers implement a controller and watch loop:
  - "Deployments" – offer scalability & rolling updates
  - "DaemonSets" – run instance of service on every cluster node
  - "StatefulSets" – used for stateful components
  - "CronJobs" – for short lived tasks that need to run at specified times

CORE KUBERNETES COMPONENTS

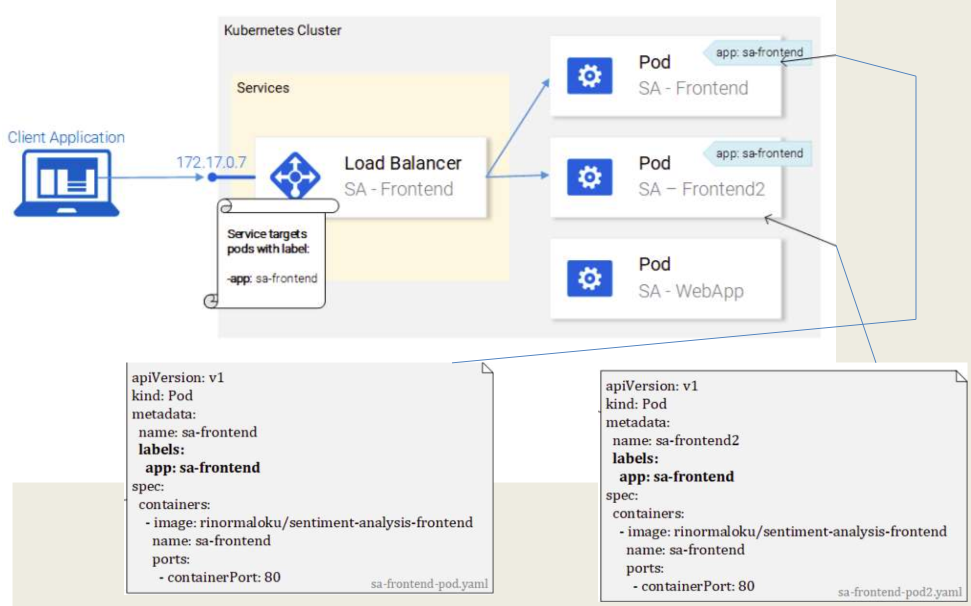
- Kubernetes DNS
- Pods
- Services

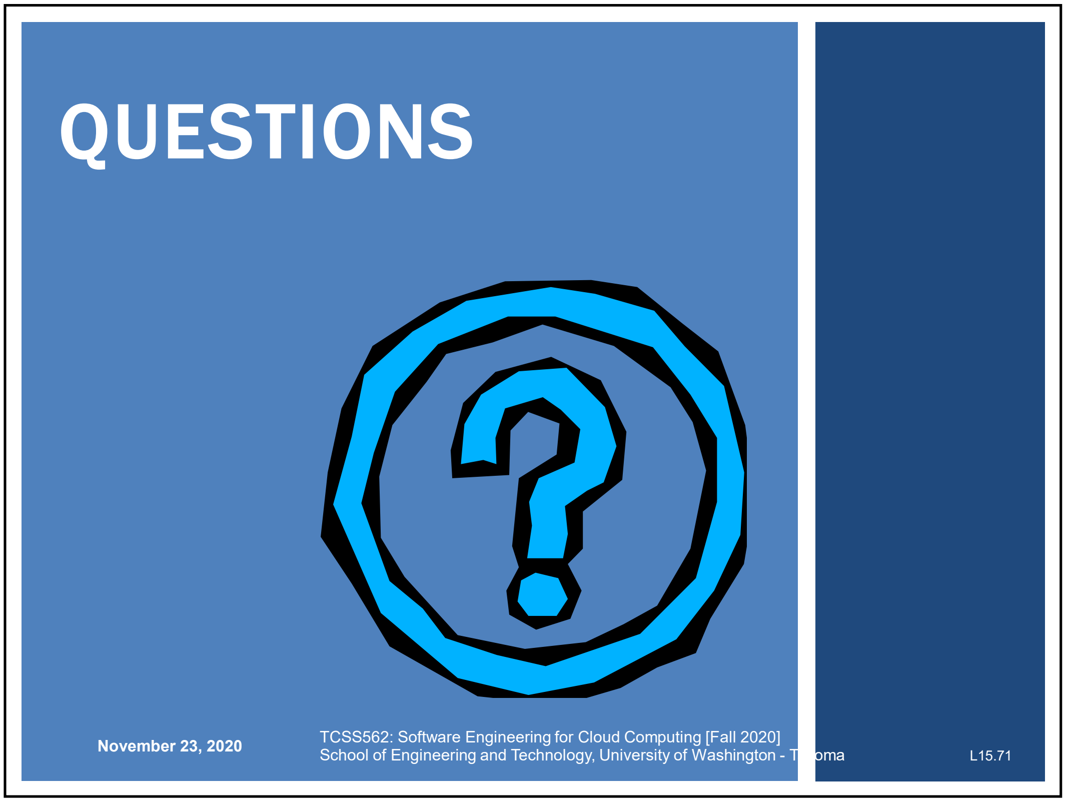
KUBERNETES "SERVICES"

- Pods managed with "Deployments" or "DameonSets" controllers are automatically replaced when they die
  - This provides resiliency for the application
- **KEY IDEA**: Pods are unreliable
- **Services** provide reliability by acting as a "GATEWAY" to pods that implement the services
  - They underlying pods can change over time
  - The services endpoints remain and are always available
- Service objects provide an abstraction layer w/ a reliable name and load balancing of requests to a set of pods

SERVICES

- Provide reliable front-end with:
  - Stable DNS name
  - IP Address
  - Port
- Services do not posses application intelligence
- No support for application-layer host and path routing
- Services have a "label selector" which is a set of lables
- Requests/traffic is only sent to Pods with matching labels
- Services only send traffic to healthy Pods
- <u>KEY IDEA</u>: Services bring stable IP addresses and DNS names to unstable Pods




QUESTIONS
November 23, 2020
TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma
L15.71