O’Reilly Live Online Training

Kubernetes in Three Weeks

Answers to Questions from Attendees

Herein are my answers to many of the ***great*** questions posted by the attendees of the Kubernetes in 3 weeks series. Some answers are filtered through my opinions and convictions. I may have missed some other points, but my intention is to share what I can. - Jonathan Johnson

Course Information

Q: How much programming knowledge is needed before learning about Kubernetes with this O'Reilly training course?

A: Very little. We will not be coding too deep into the weeds of Microservices and things. Most of the "code" we investigate are declarative Kubernetes manifests in YAML file form. These are really just configuration files. In Part 1, week two we do dive into a few different languages and review some pre-written code to accomplish some goals. However, no programming is required. Fundamentally, it’s best to just be familiar with running command-line tools in Linux and be able to understand architectural concepts that involve calling services across a network.

Q: How much knowledge of containers is required for Kubernetes?

A: Kubernetes is a container orchestrator. In order to run applications on Kubernetes, your applications must first be packaged into container images. To be a proficient engineer with Kubernetes, you should master container image building. However, you don't need to learn them in the strict order of containers first, then Kubernetes. Once you understand that a container is just a thing that runs and to Kubernetes, all containers look the same, then you can learn Kubernetes knowing that it's just a "thing" that you run. The details inside the container are not that important to Kubernetes. I have a few scenarios that show you how to build an

application in various languages into containers. Each of these scenarios takes these containers that you build and runs them on Kubernetes. This is a great place to start and we cover these in week two of Kubernetes Fundamentals in 3 weeks.

Q: Will this training be touching the sections for CKA certification?

A: There are three types of Kubernetes-related certifications that are offered by the Cloud Native Computing Foundation (CNCF). The certifications are CKAD, CKA, and CKSS for application development, administration, and security interest respectfully. The basics of these exams can be found here.

My O'Reilly online training starts with introductory topics and over the parts and weeks, the topics build on each other. My introductory topics and hands-on scenarios help teach the skills that will need to be mastered to pass the CKAD exam and some topics related to the CKA and CKSS exams. However, my topics go beyond the confines of the specifics of the exam. To train for the CKAD (application developer) certification exam I would highly recommend the O'Reilly training hosted by Benjamin Muschko called the Certified Kubernetes Application Developer Crash Course (CKAD). He teaches specifically the CKAD exam and offers Katacoda scenarios to tune your skills for passing the certification.

Many people have successfully passed the CKAD certification exam by attending these O'Reilly courses. If this is your interest, you are on a good track.

Q: What is the relationship between Docker and Kubernetes?"

A: Docker is many things. It's a company, a command-line tool, and a public registry called DockerHub. There is also a small language called Dockerfile. So Docker is many things all related to building container images and running containers. When discussing containers, I rarely use the word "docker". It's like calling a tissue a Kleenex. Sometimes brands become so strong they obscure the greater idea. There are various container runtime engines that can run any OCI container. There are several tools that can build OCI compliant containers. So really your focus should be on OCI container builder and their runtime engines, and not on the Docker brand. Most of your applications can be packaged in OCI containers.

Kubernetes is many things, including a container orchestrator. Kubernetes simultaneously manages the lifecycle of 1000s of containers across multiple machines in your data center. That data center also called a cluster can be a variety of targets such as public cloud, private clouds, hybrid cloud, on-premise, and on your laptop.

Your applications are packaged into OCI container images. Kubernetes can run your OCI containers across a cluster of machines.

Q: Will we cover Helm during this training?

A: Yes. Helm is covered in week 3 of Part 1. There is also katacoda scenario to teach how to install charts and create your own charts.

Q: Will topics on governance and orchestration be covered?

A: Governance and orchestration are covered throughout the topics on Kubernetes as this is what Kubernetes gives you. We look at orchestration from a few angles. You may want to review the architecture diagram from week 1 where we talked about the Kubernetes control plane components such as the controller manager, scheduler, and Kubelet.

Q: Will we cover ArgoCD during this training?

A: ArgoCD is a rising continuous delivery solution for DevOps and I like what they are doing. They seem to be grabbing a larger foothold in the Machine Learning community, such as with Kubeflow. While compelling I cannot cover all the CI/CD tools. There are many.

General Architecture

Q: Is the use of Kubernetes primarily or exclusively intended for microservice architectures or is it also intended for monolithic architectures?

A: At the end of the first section "Distributed Computing" of the slides in week 1, I reference a listing of six space-based architectures described by Mark Richards and Neal Ford. All of these can potentially be applied on top of Kubernetes. However, Kubernetes shines because of its ability to resist the failures of distributed computing. Therefore architectures that encourage modularity with high cohesion and low coupling are best suited for distributed computing. Microservices is at the top of "...ilities" that work with Kubernetes because of its agility, deployability, testability, scalability, fault-tolerance, and evolvability. But if you look at the other "ilities" ratings for microservices you will notice "simplicity", "cost", and "performance" are not well rated. Microservices is also an architecture that is very amenable to be placed into distilled containers. There are balances of pros and cons to weigh when moving to Kubernetes. Your entire team and CTO should be aware of these characteristics.

Q: Can you describe the difference between orchestration, containerization, and microservices?

A: Microservices is one of many architectural patterns. It's noted for its affinity for high cohesion and low coupling. Typically microservices solve one modular business feature and expose its public API as a network endpoint using some protocol like REST with JSON, gRPC, or GraphQL. Microservices can be written in a variety of languages and frameworks. Microservices tend to own their own data stores, are self-reliant, and tolerant of other connecting services that may not always be available.

Containerization, popularized by Docker, is a technique of packaging an application along with all its dependencies into a tarball called a container image. Any container runtime engine can receive this tarball, unpack its contents and run the application on a host operating system. Each application that is "containerized" is provisioned using Linux isolation techniques such as security, chroot, namespaces, and cgroups. All this isolation makes this Linux process "feel" like it's a full Linux OS, where in reality it's sharing the resources of the host with other containerized applications. Each containerized application is isolated, yet uses the same kernel, CPU, Memory, and I/O of its host machine. OCI formatted container images and runtime engines have become the standard.

Microservices and containers are two very different types of architectures. Together they work like peanut butter and chocolate and as an architect, I would always put a microservice into a container.

Orchestration kicks in when you realize that connecting more than just a few containers gets exponentially harder. Orchestrators help you manage the life cycles of multiple containers. This allows you to have resilient instances that can be observed and scaled while all connected across the same network plane. Kubernetes is an operating system that eases the complexities of distributed applications as it orchestrates swarms of containers.

Q: You mention that you wouldn't deploy a monolith on Kubernetes, I agree that the apps are big - is this approach not suitable for breaking the monolith into Microservices?

A. You can put a monolith in a container as many things can be stuffed into that pillow case. However, just because you can does not mean you should. Since Kubernetes is a distributed operating system managing numerous clones across a network, how would you clone and distribute your monolith that was not designed for distribution? You could do statefulsets and scale traffic to cloned monoliths, but this design would be completely missing the points of the advantages of distributed modular applications that leverage the wealth of CPU, memory and

I/O resources across multiple nodes. Monoliths tap just the CPU, memory and I/O of a single node.

Q: Are microservices in cloud native a good idea? Because it increases the number of instances and you end up paying more for each instance, rather than what you would have paid for a monolith application.

A: It depends. Some of this question is addressed in the previous answer. Keep in mind that not everything in a container is a microservice. For instance, Redis, RabbitMQ, Kafka, and Tensorflow all run in containers, yet are not considered microservices. All of these run well on Kubernetes. If you are designing with a microservices approach, you also may not want to run everything in a container. There are teams that run their entire customer solution on microservices, without the use of containers. Microservices, containers, and Kubernetes are three independent architectures with each having a notable cost and learning curve. Combining all three into a commercial solution typically starts paying off when you have a growing count of customers, traffic hits, business features, business units, development teams, deployment targets, system outages, data center costs, and revenue. The need for agile, fault resistant, and scalable services will drive you toward cloud native. Currently, I see Kubernetes as the key operating system for large cloud native solutions, but it can be overkill for smaller designs.

Q: Can monoliths and non-cloud native apps (stateful) benefit from moving to Kubernetes?

A: Some of this question is addressed in the previous questions. Monoliths can run in containers and can run on Kubernetes. But just because you can doesn't mean you should. Monoliths resist being distrusted and scaled and this is often the reason why they are so common as they are easier to develop. It's not the right fit to put an application architecture that resists distribution on a distributed compute infrastructure. However, it could be part of your monolith to microservices transition plan. Some of the best advice on moving from monoliths to microservices is from Sam Newman in his recent book, "Monoliths to Microservices.". Also, don't fret over stateless versus stateful applications on Kubernetes. Kubernetes is designed to run *stateless* and highly ephemeral containers to run fully *stateful* applications connected to persistent stores with unique IDs. See StatefulSets and Volumes.

Q: How do government organizations feel about this architecture?

A: I'm not sure governments have feelings. That seems like a dangerous proposition. We, the people, have feelings and I will leave it at that. Instead, I would follow the money. By architecture, I assume you are implying "cloud native". I see Kubernetes as a key operating system for cloud native solutions. Remember I had a slide on the knights of the roundtable? While the CNCF governs Kubernetes, no one really owns it. All of these knights have significant

investments in the success of Kubernetes. As an example, these "knights" are all eyeing the U.S. Pentagon $10 billion JEDI contract with Microsoft and its Azure cloud native space. This ripple effect has opened up more acceptance of the cloud which translates to containers and orchestrations systems such as Kubernetes. Aside from this elephant, there are many press announcements coming from companies like D2IQ, VMWare, Rancher, RedHat (to name just a few) that talk about government contracts that leverage Kubernetes.

Q: What are the drawbacks of Kubernetes?

A: Distributed computing is hard. Remember that Kubernetes is just a tool to help you wrangle the complexities of distributed computing. Given that here are the frictions: cost, performance, and simplicity. The learning curve and expense to raise your team's maturity model can be unreachable for some organizations. See "ilities" question above. If you do not see the benefits of distributing your customer solutions across multiple machines, then stop. Your life might be complicated enough.

Q: Is Kubernetes the only choice for running containers?

A: There are a few others. However, Kubernetes has emerged to be the de-facto standard for orchestrating multiple containers. Because Kubernetes evolved out of existing systems at Google and is now an open-source project that is not owned by any vendor or cloud company it has flourished. Many companies that were writing their own container orchestrators soon realized it's very difficult to do so; they instead refactored their offering and integrated Kubernetes as their core engines for orchestrating containers. OpenShift, Rancher, Tanzu, D2IQ, Merantis are good examples of companies and products that are now doing well after readjusting their efforts to work with Kubernetes rather than compete. Azure Container Service is now Azure Kubernetes Service (AKS). Many have replaced their Cs with Ks. Reinventing a smaller wheel is rarely a good idea.

Amazon Elastic Container Service (ECS) is a relatively less extensive solution to EKS, but ensures a tighter lock with that cloud platform. If you are looking for a lighter-weight container orchestrator, consider HashiCorp's open source Nomad.

Q: Where can I find a checklist for maturity model evaluation?

A: I don't have a public one yet. I love the ideas from the book, The Checklist Manifesto, by Atul Gawande. An open-source list where our community can adjust the questions and see anonymous responses for relative benchmarking would be helpful. Perhaps I'll publish a draft on my website soon. In the meantime take a look at the thoughtful work at Armory. https://www.armory.io/blog/stages-of-software-delivery-evolution-infographic/. What stage are you at?

Q: What is the difference between serverless and cloud native?

A: "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach." - Cloud Native Definition v1.0.

My translation is it's the target where you want to run stuff. Serverless is a term driven by the frustration that getting your stuff to run on the cloud is hard. I see serverless as a software engineering goal to make deployment, delivery, eventing, scaling, and all the other Kubernetes "...ilites" simply "boring". We will discuss Serverless in Part 2, as it's ideal for running with Kubernetes.

Q: How does Kubernetes compare with other resource managers like yarn, Mesos?

A: It depends on your needs. There is no perfect solution for distributed computing and effective resource management. Each one is going to have its community of support, maturity, and effectiveness for your needs. I find Kubernetes approachable as it has taken the hard parts of distributed computing and made it relatively easy for us. Baked and proven at Google for 10 years with millions of containers before it was released.

Q: Multi-cloud and hybrid seem to be two challenges that Kubernetes solves immediately. Do you think Kubernetes is a great leveler?

A: Sometimes when I close my eyes I dream I can fly, then I open my eyes. When we remain agnostic from a specific vendor's opinion or dependency and can keep our standards open our whole community benefits. Not all businesses want to move to the cloud. Some businesses are finding local data centers are no longer cost-effective. No one is wrong or right, and we should encourage the spectrum of choices like this. Kubernetes architecture treats the underlying nodes and resources as commodities. My idealism side would like to see secure and performant nodes offered on an exchange and my intelligent Kubernetes scheduler based on AI chooses the most economical Nodes that meet my criteria. The same for my Kubernetes control plane. However, this is science fiction and vendors will resist this type of leveling. There has been growing interest in hybrid Kubernetes clusters where some nodes are from a single cloud vendor the other nodes remain in your private data center. There are Kubernetes partner companies that can help you assemble a hybrid cloud solution.

Q: Do microservices imply a specific compute resource? (example: ServiceFabric, AKS, Container Instances)

A: Never. Microservices is an architecture pattern and should not be implied to be tied to a specific implementation or resource. Not even containers.

Q: I could bake the security concerns in the Kubernetes deployment via namespaces and network policies rather than have something like JWT based access control. Is it a good idea to have everything relying on Kubernetes?

A: It depends. Overall I think we tend to put too much extra logic and dependencies in the core business features we deliver to our customers. Let's take a Spring Boot application. There are tons of extra features in Spring Boot to add TLS support, metrics, logging, health checks, and the like. All these things have very little to do with the core business feature the user wants. Say your user wants to buy a concert ticket. What does purchasing a concert ticket have to do with TLS connections or logging? I often repeat the importance of high cohesion, modularity, "do one thing and do it well", and container distillation. With Kubernetes, and even with a good mesh added on top, many of the features you thought were a good idea to put into service, start to look like an anti-pattern. Let's say you need to be security compliant for delivering a regulated and certified software solution. One of those requirements is often mutual TLS communication between all services, even within a private cluster. As you know containers embrace polyglot. Are you willing to code in all the different languages TLS for all communications as well as manage all the security keys, storage, and rotation? That's a bunch of work, unrelated to delivering true value to your customer - e.g. does my patient have cancer or not? With Istio mesh there is a switch that secures mutual TLS communication between all services, and it will manage all your keys all outside of your code. So it depends, but I would say be vigilant about high cohesion. When you add a dependency, make sure you add it in the right place.

Q: We're moving from Cloud Foundry to Kubernetes (from a true PaaS to a container orchestrator platform) given some of the complexities of BOSH and other Cloud Foundry components. However, there are now developments happening where we're seeing both Cloud Foundry and Kubernetes are playing together - KubeCFG, Gluon, Eirini, etc ... What exactly is the right optimum environment for a cloud native ecosystem?

A: Pivotal's Cloud Foundry is now part of VMWare. There is a trend where PCS solutions are moving toward PKE. Notice the C went to a K. Amazon has ECS and EKS, same morphing going on. Rancher dumped the Cattle scheduler in favor of adopting Kubernetes scheduler as

the core for their solutions. In these transitions, you will also see hybrid solutions. Just be careful of vendor lock-ins, but don't be afraid to buy in with your vendor when the costs make sense. As an architect, I like to ensure safety to pivot to other solutions when my CTO says, we need to save money, let's move to X. Open-source Kubernetes is agnostic to vendors.

Q: Not a question, but more of a comment. ECS does support the idea of Pods via Tasks. A task is a grouping of one or more containers, and an ECS service is a replicated/balanced group of tasks

A: Yes, there are similarities and differences with this specific vendor solution.

Q: What is that space-based arch that you mentioned in the slide? How is it different from microservices?

A: Microservice is one of the 6 space-based architectures that Mark Richards and Neal Ford have outlined in their book. See Chapter 15. Space-Based Architecture Style.

Q: How would Kubernetes architecture look like for Continuity of Operations (COOP)/Disaster Recovery (DR) site vs ENCLAVE (Primary)?

A: I have not explored these ideas enough to elaborate. Good question though.

Q: When you say low coupling that generally refers to external actors accessing the container's microservice interfaces. How would you classify microservices in a container that exclusively consumes from other microservice APIs - for example pipeline services?

A: The code for a microservice should have no knowledge about how it's packaged nor should it have any coupling to the ideas that it’s running in a container or orchestrated with Kubernetes in a cluster.

Q: What is the difference between ambassador and adapter patterns?

A: Ambassador pattern is typically wrangling many disparate and dynamic backs ends. Adapter is simply a translator, typically 1 to 1.

Applications on Kubernetes

Q: Can we use Hadoop, Samza, Flink on Kubernetes?

A: Yes, if it runs in a container it can run on Kubernetes. However, just because it can, does not mean you should. "Sure, no problem", says DevOps. "It depends…", says the architect. There is a trend where people are finding Kafka and Spark on Kubernetes is meeting their data processing and stream processing requirements. Perhaps this blog will shed some light.

Q: What about running systems like Oracle's Peoplesoft, have you heard of someone running this kind of system on Kubernetes?

A: I worked with a company exploring the idea of moving a Laboratory Management Information System (LIMS) to Kubernetes. There are many types of enterprise systems like this such as human resource management systems (HRMS), Financial Management Solutions (FMS), supply chain management (SCM), customer relationship management (CRM), and enterprise performance management (EPM) that all can run on Kubernetes. Really it's up to the architects and business owners to decide whether it's a good idea. Therein lies the hardest thing about Kubernetes, are the costs and benefits enough to justify using Kubernetes as the underlying architecture for your system? How will the customers benefit from that move? I'm not sure what Oracle is thinking, but many companies are wrestling with the "is it worth it" question. It's an easier decision with greenfield code, and much riskier with brownfield and monolithic architectures.

Q: Is the Kubernetes dashboard something you install additionally or is it something that comes with Kubernetes? How do you view the Kubernetes dashboard?

A: It depends on your Kubernetes cluster, but typically there is a dashboard running as a Pod. On Minikube typing the command "minikube dashboard" will open your default browser to the dashboard landing page. On MicroK8s its "microk8s dashboard-proxy". On my Katacoda scenarios read the instructions at the bottom of step 1 or type "token.sh" at the command prompt. For high sensitivity production clusters administrators will often forgo adding a dashboard to reduce the potential attack vectors: Lessons from the Cryptojacking Attack at Tesla.

Q: What's the difference between running RabbitMQ on Kubernetes and running RabbitMQ on a cluster natively?

A: There are many ways to set up RabbitMQ and you certainly do not need Kubernetes. However, remember in Week 1 we covered the 16 features that Kubernetes offers that you never want to code? e.g. scaling, high availability, resilience, scheduling, etc. When RabbitMQ, or anything for that matter, runs on Kubernetes, you can start seeing how Kubernetes make these systems behave better when the load is distributed. Often, there are public Helm charts that the authors of these systems have written for you. These Helm charts are deployed to Kubernetes for high availability configurations. The tweaks and context for the Helm charts can be adjusted and stored in your version control repositories as infrastructure-as-code.

Q: You talked about CI/CD. There are ArgoCD building and deploying apps on Kubernetes. Are there similar products that do the same thing?

A: There is a boatload of CI/CD solutions that run on Kubernetes. The CNCF has these tiles to help you navigate the decision overload, found here. In part 2 we will cover running CI/CD pipelines on Kubernetes. Consider Tekton to help lay the first row of bricks.

Q: What are the options if I want to run a system like Storm or Heron on k8s. Those platforms want to handle similar features that Kubernetes can.

A: It depends on your needs. Herons, from Twitter, focuses on distributed data stream processing. Storm focuses on stream processing. Kubernetes is not the only way to do distributed computing, but its focus is not limited to just data stream processing. Take a look at the CNCF tiles on cloud native stream processing solutions. There are many. Both Heron and Storm can run in containers on Kubernetes and take advantage of what Kubernetes does well. "Do One Thing and Do It Well" is a Unix philosophy that culturally extends to Kubernetes. Why not delegate the "...ilities" of distributed computing to a system (such as Kubernetes), then those things that run on the operating system can concentrate on what they do well, such as data processing, streaming, CI/CD, messaging, databases, artificial intelligence. Etc. Here is a Helm chart for Storm on Kubernetes.

Q: Can we use Kubernetes as some sort of internal infrastructure provider?

A: Yes, I often recommend running your CI/CD pipelines and testing on Kubernetes. In Part 2 we will cover this topic. Before moving your applications to Kubernetes in production, start with automating your operations with code first on Kubernetes. This is a transition technique to introduce your team to the benefits and difficulties of distributed computing with Kubernetes.

Helm

Q: Why do you sometimes use Helm instead of kubectl?

A: Kubectl allows you to manipulate an object or a group of objects atomically. Sometimes imperatively using direct commands, and sometimes by submitting one or a group of YAML files. Helm also allows you to update a cluster, but it's restricted to just using named and versioned packages called charts. Helm allows you to inject contextual parameters based on the installation target when the chart is installed. Typically you choose to use Helm when you have a chart of YAML manifests that constitute a complete and installable solution set. For instance, you could install Kafka with a series of kubectl commands and or a single bundled YAML file. You can also install Kafka with a single Helm install command. Helm is a package manager for Kubernetes like Brew is to MacOS. Kubectl is for atomic admin commands.

Q: Is each Helm chart deployed to a namespace?

A: That is the most typical way and easiest to manage and understand by people using the chart. Helm charts are collections of Kubernetes manifests in YAML form. They define a collection of objects that require a namespace declaration. The command-line option for helm accepts a --namespace parameter. The namespace must exist before the install of a namespace can occur. You can deploy the same chart to multiple namespaces as long as the installation names are unique. The YAMLs can also have namespaces declared and they can differ between resources in the same chart. However, mixing namespaces in a single chart can get complicated so be careful.

Kubernetes General

Q: Is Kubernetes a container as well as an OS/toolset to manage containers?

A: Kubernetes is not a container, but a set of tools to help manage a distributed OS container orchestrator for multiple containers across a cluster of servers. The Kubernetes control plane components are a collection of containers that ensure the Kubernetes cluster is healthy and standing up balanced.

Q: Do we need a public cloud to deploy Kubernetes or can we experiment on a home pc with 8/16 cores?

A: There are many targets where you can install or get access to a Kubernetes cluster. Make sure there are a few cluster targets that are readily reachable in your professional toolbox. The scenario I provide gives you quick access to a free and lightweight cluster that is always available for sandbox experimentation and learning. Next, be sure to install a lightweight Kubernetes implementation on your laptop. Each cloud provider also gives access to Kubernetes clusters on their systems. Your company may have an existing account, so see if you can use that to create a personal cluster for experimentation. Google offers free credit for learning with a new cluster. Just be careful with the cloud solutions and spinning up a cluster attached to a credit card as the money meter will be running if you forget to shut down the cluster when you are not using it.

Q: My understanding is that OpenShift uses Kubernetes, so it's RedHat's child built on top of Kubernetes?

A: OpenShift is a wrapper around Kubernetes. OpenShift is RedHat's commercially supported implementation of open-source Kubernetes. Like many commercially supported versions, they have their opinionated set of preferred solutions to ease your adoption and use of Kubernetes. They also offer a ton of code and support to the upstream Kubernetes source code. Just upstream of OpenShift is OKD which is the open-source version of OpenShift. The Upstream of OKD is Kubernetes. Whenever OpenShift wants to be open source yet cannot get into Kubernetes will end up in OKD. RedHat was the first company to understand that writing and container orchestrating distributed operating systems is really hard to do. When they saw Kubernetes being released they refactored OpenShift version 2 and integrated Kubernetes into its core engine. OpenShift version 3 was the merging wrapping OpenShift around Kubernetes. At the same time, CoreOS had a product called Tectonic. RedHat bought CoreOS and RedHat version 4 is the merging of all the great ideas from CoreOS into OpenShift. Soon after IBM bought RedHat for $34 Billion - the largest software purchase so far in our industry. This continues to be a success story. VMWare with Tanzu, Rancher, and a few others see this too.

Q: Do Kubernetes and OpenShift complement each other in the real world?

A: See answer above. Learn Kubernetes first and remain independent of any vendor. Later you can always see how each vendor helps you to make it easier in their own unique ways.

Q: Can I seamlessly port my applications from OpenShift to Amazon EKS?
A: Both OpenShift and EKS are just Kubernetes underneath, and part of the Kubernetes architecture is ensuring there are levels of indirection to keep agnostic to a specific target. For instance, many of the same Helm charts I use to install applications on Minikube can be used to deploy to the Katacoda playground or to GKS. There are some environmental contexts like the

number of pods or locations of persistent volumes often can tweak. So relatively it should not be hard. The difficulty lies in the number of couplings to specific OpenShift only features and are not generic to Kubernetes. Moving to Amazon ECS would not be as seamless. Once you move to a specific cloud vendor's Kubernetes as a Service (KaaS) they will offer many services like observability for day-2 operations. You can buy those conveniences with the risk of lock-in, find another 3rd party to add solutions on top of Kubernetes, or you can build it on Kubernetes generically.

Q: What is the best production-ready distribution for on-prem (bare metal) with licensing follows Apache 2? I don't see many as most on-prem are commercial and those Apache-2 licenses are just single node learning type distribution.

A: There are many vendors that would like to work with you for standing up on-prem solutions in your private data center. There are over 90 certified vendors: https://www.cncf.io/certification/software-conformance/. And here is a list of each vendor certification status (notice the Type column to find on-premises/distributed): https://preview.tinyurl.com/jw7mqt8. For experimentation, you can build your own with Kubeadm. Or, to build some confidence and street credit, Kubernetes the Hard Way, from Kelsey Hightower.

Q: Do Kubernetes differ between local laptop, on-premises, and cloud installations? How different is it running Kubernetes in different cloud vendors such as AWS (EKS), Azure (AKS), GCP (GKS), IBM/RedHat (OpenShift), VMWare (PKS)?

A: It's a pleasant surprise, that it does not differ as much as you might expect. This is why I encourage you to learn on Katacoda and on your local laptop, but when you are ready to shell out your credit card for larger clusters much of what you already learned works on the cloud. The command-line tools Kubectl, Helm, and other projects will all be the same. The dashboard looks the same. There is always *default* and a *kube-system* namespace. The way to create Pods, delete, update, and put services in front of them is the same. The way to scale and the way it performs fault tolerance is the same. All of the concepts are the same. The reason is Kubernetes was designed to be agnostic from the commodities of every changing hardware that it will need to run on. There are layers of indirection and conceptual hurdles you have to grasp to achieve this isolation, but it’s worth it. Probably one of the best examples of these indirection layers is the connection of your services to persistent volumes outside the cluster. To your application, it's just an I/O mount. Kubernetes provides some magic to connect your mount to some physical hardware. Operating systems have device drivers, Kubernetes is no different when connecting to resources. When you are exploring different vendors they will entice you with many extra services and conveniences. You may want to take advantage of what they offer

as it's a buy vs build decision. Just be deliberate about where your coupling to vendor dependencies.

Q: What are your thoughts about cloud-hosted Kubernetes environments? Where, the cloud provider manages the underlying infra?

A: As you might tell from my answers, I love to remain as open and vendor-agnostic as possible. However, I also know how deceivingly hard it is to build and maintain your own software and solutions. If you have the budget, buy it. What you buy will be vastly better than what you build, usually. For instance, there is no way most companies can approach the level of security and monitoring that AWS provides for its resources. If you believe you could create an on-premises data center that is just as reliable and performant as a cloud vendor for the same amount of money, you are most likely mistaken. Most people simply cannot hire, train, and keep the experts it takes to run these cloud data centers. For cloud systems, pick a vendor and buy their KaaS system. For hybrid or on-premises you should also pick a vendor to work on making your local cluster secure and performant. If you build (not buy) your own Kubernetes cluster on-prem, your customers may ask you for some type of assurance/insurance from a neutral Kubernetes approved vendor. There are over 90 certified vendors: https://www.cncf.io/certification/software-conformance/. Here is a list of each vendor certification status (notice the Type column to find on-premises/distributed): https://preview.tinyurl.com/jw7mqt8

Q: What is kubeconfig? What are the ways to authenticate/authorize to a Kubernetes cluster?

A: Kubeconfig refers to a specific type of manifest file that holds the context for administration access to one or more clusters. This file resides on the client machine where tools such as kubectl, helm and other applications will reference this context information to gain access to a cluster. The default path where this file is located is $HOME/.kube/config. A similarly named environment variable $KUBECONFIG holds a list of paths to one or more kubeconfig typed files.

The command "kubectl config get-contexts" will list the contexts of the different clusters you can access with kubectl, and other client tools. The help command "kubectl config --help" will list the actions to maintain your cluster contexts.

Q: Can you explain creating kubeconfig.yaml

A: The kubeconfig.yaml file, located in your home directory, stores the context information for connections to various clusters. Kubectl, Helm, and any other command-line tool that needs access to the Kubernetes Client API server on any cluster needs information on how to securely connect to one or more clusters. Try 'kubectl config' and 'kubectl cluster-info' More on this file is

here. You would not typically create this file from scratch. Whatever cluster you connect to typically has a tool to generate this file for you. Minikube creates the file and EKS, AKS, GKS all have command-line tools that offer a way to generate the kubeconfig.yaml automatically with the appropriate context such as user access, URLs, and tokens. If you know all the access and secrets to the cluster of interest use can use a series of set and use commands like this:

```
$ kubectl config set-credentials myself --username=admin --password=secret
$ kubectl config set-cluster local-server --server=http://localhost:8080
$ kubectl config set-context default-context --cluster=local-server --user=myself
$ kubectl config use-context default-context
$ kubectl config set contexts.default-context.namespace the-right-prefix
$ kubectl config view
```

Q: Is GitLab a certified K8S container orchestrator?

A: No. GitLab is a DevOps platform. Ideal for GitOps and DevOps practices such as delivery pipelines where you can build up sophisticated continuous integration and continuous delivery pipelines. Being open-source you can run an entire GitLab stack ON Kubernetes. Kubernetes is ideal for running any kind of DevOps solution and provides a resilient and secure foundation for scaling your parallel pipelines. The pipelines you develop will often deliver applications TO Kubernetes clusters. You can have pipelines manage the life cycles of multiple Kubernetes clusters and their applications.

Q: Does Kubernetes support Distributed File Systems (like Hadoops HDFS)?

A: Yes. Kubernetes is a distributed operating system so naturally, there are several solutions that can stand up a highly available distributed file system on Kubernetes. The CNCF has collected many of these solutions into the group Runtime - Cloud Native Storage. Some of the popular ones are supported by Ceph, RedHat's Cluster, OpenEBS, and Minio to name a few. Whether your storage is distributed or not there is a Common Storage Interface (CSI) and many OSS solutions and vendors provide drivers for file storage behind the CSI API. The list of drivers is large.

Q: Our company is starting on its Kubernetes journey and we need to run a hybrid architecture of VMs and containers in Kubernetes. Any pointers?

A: Hybrid architectures with Kubernetes offer unique challenges and it's best to partner with solution providers that have done this work with clients before. This list might be a good starting place.

Kubernetes Control Plane

Q: Is Kubernetes written in Go?

A: Yes, Google has standardized on the Go language for the control plan of the Borg project and most of its containers. The early version of Kubernetes was written in Java but was soon converted to Go. Google created the Go language because they wanted something efficient in containers, yet easier to write than C++.

By no means does this imply or require your containers and application running on Kubernetes to be written in Go. After all, containers embrace polyglot. All of the source code for the Kubernetes control plane components like api-server, schedule, controller manager, etcd, kubelet are written in Go and available for all of you to see and improve on Github.

Q: Does the control plane need to be configured by us?

A: Typically no. As a developer how often do you dive into the control plane of Android, Windows, Linux, or any other operating system? Not often, unless you need to write a driver or add a new tool. You can if you are interested and the Kubernetes design shines by how easy it is to adjust, extend, and even replace behaviors. As a developer, the more time you spend tweaking Kubernetes, the less time your customers will get your attention. Kubernetes as a Service (KaaS) system offers you the best options to just purchase Kubernetes as a managed service so you can concentrate on delivering solutions to your customers (build vs buy). However, Certified Kubernetes Administer (CKA) types of personas will be interested in configuring the control plane. For large clusters, you will need CKA roles even if your purchase a KaaS system.

Q: What does the kubelet do?

A: A Kubeless is a daemon process that runs on each Node in a Kubernetes cluster. Each Kubelet on every Node receives instructions from the kube-scheduler to manage the life cycles of the Pods. The Kubelet manages the Pod create, update, read, delete (CRUD). Each Pod has 1 or more containers. As the Kubelet is managing the Pods, it connects to the container runtime engine, also on each Node, and communities through the container runtime interface (CRI) to manage the life cycles of the containers as the Pods are being CRUDed. Each Kubelet reports back to the Kubernetes control plane the health and state of the Pods.

Q: I would like to understand DockerShim and what will be the effect of its removal from Kubelet.

A: Container runtime engines manage the lifecycle of containers. Kubernetes purposefully keeps itself completely isolated from the details of containers. Kubernetes delegates all the container management to each container runtime engine running on the host of each Kubernetes node. The Kubernetes's Kublet can agnostically ask any container Container runtime engine to manage a container through the Container Runtime Interface (CRI). as long as a container runtime engine implements the CRI interface, then it can manage all the containers on a Node. You can even have different container runtime engines running on different Nodes and Kuberentes will not care.

The problems with the Docker container runtime engine was it did not implement a CRI interface. So a project called DockerShim must be run next to the Docker engine. The shim is an adapter that implements the CRI interface and also connects to the Docker engine. Having this extra component and maintaining the updates to CRI and the Docker engine has proved to be difficult. With the ruse of OCI based containers, there are now several evengines that directly implement the OCI interface and can run any OCI image. These engines offer better security and optimization over the Docker engine. For instance RehHat's OpenShifjust t relies on the cri-o container runtime engine. Overall, the containers that run on Linux have broader capability than what Docker offers.

DockerShim with the docker engine has been dropped as the default engine. Moving forward, the preferred CRI engine is called containerd. OCI engines still can run docker images but you should be leaning towards OCI based containers.

Because Kubernetes is not bothered by the implementation of the container runtime engine, the decision to deprecate DockerShim, is not a big deal. It's real;y a great improvement and a testament to Kuberentes ability to evolve. The CRI interface opens doors to other types of application isolation with containers or with other ideas such as unikernels. Experimentation is underway to package applications using WebAssembly/WASI.

The real reason why this appears to be big news is many people continue to refer to containers as Docker or Docker containers. People use brand names for generic terms all the time. Don't call tissues as Kleenexes, searching as Googling, and do not call containers or container images as Docker. In your mind, deprecate the term Docker and instead just think of the architecture as containers with the standards:

- OCI (Open Container Initiative)
- CNI (Container Network Interface)
- Kubernetes CRI (Container Runtime Initiative)

More on the deprecation of DockerShim is in this article.

Q: When I need to preserve my data, does my data live on because it is in etcd?

A: Application data does not live in the etcd found in the Kubernetes control plane. The etcd at the Kubernetes control plane level is just for storing the declared and status state of all the cluster resources and objects. Your regular applications on Kubernetes do not have access to the etcd for the control plane.

For your applications, you can choose to use etcd as a key/value distributed data store (there are other options too). You can certainly install etcd on Kubernetes just like any other application. There are good Helm charts and Operators that will help you install and manage etcd on Kubernetes.

Q: Kubernetes necessitates a lot of monitoring. Can artificial intelligence be implemented to monitor and act on it?

A: Yes, really distributed computing and cluster environments require lots of ***observability***. Monitoring is just looking at metrics, but there are also logs and tracing. Not to mention dashboards and alerting rules and channels. Observability is a significant part of Day-2 operations and an important part of your maturity model measurement. Because observability includes gathering lots of data, that implies AI can and should be applied to it. For very large clusters applying an AI-based scheduler would be amazing. Also, Pod and Node scaling can be done, before backpressure problems arise. I have a feeling large production Kubernetes clusters will be led by AI Operators freeing operations from many of the mundane decisions. Cue the videos of the Tesla drivers asleep at the wheel.

Q: Why do you call Kubernetes an Operating System?

A: Obviously it's not an operating system in the classic sense. Since computers started we have been trained to think an operating system controls a specific unit of hardware. On the hardware are resources to be managed, or orchestrated. Those resources can generally be grouped into CPU, memory, and I/O. A Kubernetes cluster is a collection of Nodes. Each node has a host operating system that efficiently runs containers and manages the CPU, memory, and I/O for the containers on each node. However, you as a developer care very little about all those commodity nodes. When you declare to Kubernetes I want 100 replicated microservices, then Kubernetes orchestrates all those containers in Pods across all the appropriate Nodes. To you, Kubernetes is managing the CPU, memory, and I/O details, therefore it's an operating system. It's a viewpoint I want you to consider to help understand the power of Kubernetes. I'm not alone

in this thinking as D2IQ calls their container orchestration product the Data Center Operating System (DC/OS).

Q: Does the scheduler take into account geographical distance from resources to service request location?

A: One important deployment issue about Kubernetes is you want the services and data close to your users. Having a cluster running in the UK for someone in Singapore will not provide optimum performance. People do run replicated Kubernetes clusters across regions. Your cloud vendor KaaS solution will have recommendations for running Kubernetes across multiple regions. Also, have a look at this article, Running in multiple zones. There is also a notion of Federated clusters.

Q: Can we have more than one master node?

A: Yes, and it's a requirement for distributed computing and a hardened production cluster where you want to claim high availability. If you purchase Kubernetes as a Service (KaaS) from a cloud vendor they will ensure there are at least 3 and separated. For instance, Amazon's EKS ensures there are 3 masters each in their own availability zones. When you purchase these Kubernetes services they will offer a service level agreement to keep these healthy (uptime, performance, monitoring, security patches, etc). When you run Kubernetes on a laptop or in experimental clusters there most likely will be a single master when efficiency and simplicity will rule over high availability. This is an example of how Kubernetes can flex its architecture for various target needs.

Q: Does Kubernetes have a central monitoring system with which I can visualize the current utilization?

A: Kubernetes is an extensible container-orchestration system, not a Platform as a Service (PaaS). It's not a silver bullet with everything installed and running by default. Can you observe log, tracing, and metrics? Yes, absolutely, but it's up to you to install what you want. For Logging, a common Helm chart to install is the ElasticSearch+FluentD+Kibana (EFK) solution. You can try it here, Kubernetes Observability: Logging. For information in tracing you might want to start with these scenarios on OpenTracing. Lastly, a common installation for observing metrics is the Prometheus stack, here is a Prometheus scenario on OpenShift. This is a large Day-2 topic to cover in a short answer. I will be covering Observability in Part 2 and more scenarios will be available.

Q: If you have 2 nodes only, is it better to have 2 x Master nodes or 1 x Master node?

A: If you only have 2 nodes for your cluster, by definition you are not talking about wanting high availability. A minimal high availability Kubernetes set up just for the control plane would be 2 master servers and 3 etcd servers. Each one is isolated in different "zones". That is a total of 5 servers, just for the control plane. Then you need at least 2 servers for your worker nodes. So a minimal high availability cluster would be 7 servers. See how it adds up quickly? Some administrators allocate just 3 servers with 3 masters and etcd on each master. If you lose one of those nodes, you will lose a master and an etcd at the same time. To me, that sounds like a bad idea, but it's described that way in the Kubernetes documentation and I think some cloud vendors are comfortable with that as well. So back to your question, with two nodes I would put the control plane and etcd on one node and all the other worker stuff on the other node. This is what Katacoda does when you run Kubernetes. Try `kubectl get nodes` and you will see.

Q: Do the multiple layers of "Master Node" indicate redundant instances? If so, what is the inter-communication between those nodes to ensure coherent instruction from (presumably) multiple kubectl commands?

A: Yes, the diagram I presented shows multiple masters and I visually stacked them. They rely on the source of truth for all states to be in etcd. If you purchase Kubernetes as a service from a cloud vendor they will manage these nodes and the high availability (hopefully). More about multiple master management is here.

Q: In the diagram, etcd was shown outside the master node box. Is etcd installed on the master node?

A: I'm not sure what the answer to that debate is. Some experts concerned about high availability have said the masters and etcd nodes should be on separate machines all in separate zones. The most costly proposal. Some have said just 3 master nodes with etcd running on each is just fine. In my diagram, I separated them to emphasize they are modular and independent of each other. In the Kubernetes documentation, the diagram shows etcd on the same servers. To me, it looks like a pure choice between risk and cost. Cloud vendors may be inclined to lump etcd into the master nodes to maximize their profits.

Q: Would you compare/contrast network vs application load balancers in a Kubernetes Context?

A: There are two types of load balancing events that occur with traffic between your application Pods. The first one is through your traditional load balancer. You may have a hardware

appliance on-prem or a cloud load balancer service. The traffic from the load balancer needs to coordinate with an ingress controller running in your cluster. There are many ingress controllers that run as a container in Pods in Kubernetes such as Traefik, HAProxy, Gloo, Istio, and Nginx. There are over a dozen ingress controllers listed here. These controllers will coordinate with your load balancing appliance or cloud load balancer. Once the controller is established there is a declarative manifest called "kind: ingress" where you can apply rules that map public URLs to services within your cluster. The second route of load balancing happens when traffic needs to be routed from a service to an appropriate node (round-robin). The "kind: service" rule will define the link between the named services and all the associated Pods (replicas). From there the kube-proxy control plane component on every node in the cluster will accept traffic and route it to the appropriate Pod. Kubernetes is very extendable and there is more a network engineer can do to tweak default Kubernetes networking, specifically for performance, as introduced in Richard Li's article Load balancing strategies in Kubernetes: L4 round robin, L7 round robin, ring hash, and more.

Q: Sometimes the object API version is moved from beta to stable. How to find which object controller version to use?

A: You can look at the documentation for all the resources, here is the information for version 1.18. Make sure the documentation matches your Kubernetes version. The scenario Kubernetes Fundamentals: Kubernetes API explores the version information in step 6.

Q: In the scenario "First App" is the URL to curl the request to the hello service private?

A: In this scenario, in step 2 you started the container in a Pod, but without a service there it lacks a way to access it. So, in step 3 you ran an "expose" command that created a service to front the Pod. Services can be declared with a few different types. In this case, the service type is NodePort. NodePort exposes a specific port on the master node of the cluster and we applied a "patch" command to set the port to a specific number, 31001. Katacoda has a special URL scheme so you can get to the exposed NodePorts. The scheme is https://[your Katacoda scenario instance id][NodePort][katacoda farm]/. NodePorts are a convenient testing method to access services without elaborate ingressing rules and security. For production, an ingress rule should be established between the traffic from a public DNS to a load balancer and finally to your service, all with TLS/SSL. To learn the basics about ingressing try this scenario "Create Kubernetes Ingress Routing".

Q: Is there a quick way to do A/B testing in this env?

A: Yes. There are a few ways to achieve A/B routing rules. The ingress object in Kubernetes allows you to apply routing rules based on HTTP headers. You can achieve many routing

decisions in these ingress rules. However, if you are wanting more involved rules or other types of deployments such as green/blue, canary deploys, traffic shifting, etc then adding a mesh on top of Kubernetes may be the way to go. I will cover Meshing rules with Kubernetes in Part 2.

Q: Are you doing a module on securing Kubernetes?

A: Perhaps, this series is oriented to software developers to help them understand how to move their applications to Kubernetes. There is a different persona related to Kubernetes administration. Rather than just security, I would like to ensure it's widened to "hardening" Kubernetes clusters related to what a Site Reliability Engineer would care about. Security is just a slice from that pie. I will cover NetworkPolicy in week 3 and Role Based Access Control (RBAC) when we talk about the Operator Pattern. I will also cover some security related to meshing in Part 2.

Q: I'd like to know more about the latency of routing user requests to pods for asynchronous real-time architectures.

A: I presume this question comes from the concern about adding latency performance to your application when moving it to Kubernetes. Indeed when you decide to move to a distributed architecture then latency is added to your cons list as an architect. Each time you call a service it’s a hop and there are quantifiable milliseconds delay added to your customer's experience. Different Kubernetes clusters and underlying data centers will vary on those values. If this is your primary concern and nothing else can be done to reduce response time, then don't consider moving to a distributed architecture, Kubernetes, or anything else. Keep in mind that distributed architecture can address perceived performance for the user in other ways. Because it can efficiently scale on the fly, more services can be made available to handle high traffic loads. You can add the observability pattern of tracing to also get an understanding of the bottlenecks and make appropriate adjustments to the services, resources, or scaling based on the nature of the transition in question. Tracing systems like OpenTracing, Yeager, Zipkin, or commercial solutions can run on Kubernetes and provide data for engineers to take action to optimize.

Q: How can I distinguish between if saturation of resources happened at the Kubernetes level or at the underlying machine level? Let's say I read a file. Will that be a read system calls to the underlying OS directly or it calls some API in Kubernetes framework and Kubernetes translates it to system calls?

A: The application inside a container has no dependencies or knowledge that it’s running in a container. Moreover, your application and container should have no dependencies or knowledge

that it's running on a Kubernetes cluster. That kind of coupling would be horrible. When your application makes a call, such as a system call, it's like any other Linux application. Remember that containers are really just isolated Linux processes, so any call they make is right to the kernel that the container is hosted on. Kubernetes is not involved in that communication. Kubernetes network gets involved at the routing table level and may do things like restrict or open channels between virtual IPs in the routing table, but that is all done at the networking layer. In the end, your application is still just making direct calls to Linux and should never worry about the outside universe. Most of the observability tools will be watching the metrics of process performance, memory/CPU consumptions, and traffic flow.

Q: Does a real person get associated with a Role? Or this is just for container-to-resource access?

A: The association of a role is to a ServiceAccount, not a person or a persons account. A ServiceAccount is not like a typical acount for a person such as with an id and password. Roles and ClusterRoles are not directly associated with people. A Role is bound to a ServiceAccount as defined in a RoleBinding. You can have a pool of roles and the RoleBinding can associate one or more roles to a single account. Multiple accounts can bound to the same roles. There are Roles which are associated permissions for objects scoped to an namespace. There are ClusterRoles that are associated permissions for objects scoped to the cluster. When a person uses a tool like kubectl they provide access credentials or a key. That access key is associated with a ServiceAccount. The ServicesAccount is bound to one or more Roles. Pods can also be associated with ServiceAccounts to open and restrict an application's access to Kubernetes objects. The gory details are here.

Q: I was quite sure that unless you specify a service account for a Pod then by default they do not have access to the Kubernetes API. Am I right?

A: It depends on the default service account roles for the namespace where the Pod is started. From the Kubernetes docs, when you create a Pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace. What's the default service account associated with the namespace? That all depends on what created the namespace, the cluster default service account, the cluster provider (Minikube or GKE?) and the Kubernetes version. From a safety perspective, I would assume that until you specify the rules, everything is open. I think in most cases you are right, the Pod will not have access to the Kubernetes API, but I would not sleep well until I applied a specific declaration of intent.

Q: What are your thoughts on Microsoft Hyper-V "containers"?

A: Mixing terms and technology here. Containers are run on container runtime engines. Hypervisors, like Hyper-V, run virtual machines, not containers. For any hypervisor, their notion

of a "container" is really a machine image. While container runtime engines (running containers) and hypervisors (running VMs) have overlapping conceptual models, they are significantly different with different goals. When possible I avoid directly working with machine images (VMs) as they are large, slower, consume more resources, and difficult to rapidly scale for a distributed system. VM will, however, guarantee you more isolation which can translate into better security. However, unless you are running Kubernetes raw on bare metal, all of your Nodes will be Virtual machines. On EKS they will be EC2s, on-prem they will often be VMWare nodes in your datacenter. On those VM Nodes there will be a host operating system that will know how to efficiently run containers. Take a look at how Container Linux is an operating system that efficiently runs containers that you can install on all your Worker Nodes. Your worker Nodes in your cluster are the VMs.

Q: Is there a development process to develop/change Kubernetes infrastructure?

A: Yes. Follow the notions of infrastructure as code and a high maturity CI/CD and testing pipeline. Personally, I would run my pipeline right on Kubernetes. I will cover this in Part 2 training. *The Phoenix Project*, a novel by Gene Kim, George Spafford, and Kevin Behr, is a fun book to help you visualize the development process you need. How else, in 2014, can Joe Beda from Google declare "Google starts over 2 billion containers per week (3k per sec)".

Q: Have you had experience with OpenShift. If yes, can you tell us about your experience?

A: I was a customer of CoreOS Tectonic. Then I was a customer of RedHat when they purchased CoreOS. Then I was a customer of IBM when they bought RedHat. Phew. In the transition, I only use OpenShift a bit. What struck me the most was the number of contributions both the CoreOS and RedHat teams where they continue add to upstream Kubernetes. If you want to get started with OpenShift, try the installation for Minishift. Keep in mind the upstream open-source version of OpenShift is call OKD. Upstream of OKD is Kubernetes. RedHat is a huge contributor to upstream Kubernetes and as a knight at the roundtable, they have contributed their share. RedHat has good documentation, tutorials, and many free events on learning Kubernetes. Burr Sutter (RedHat) and Steven Augustus (VMWare) have been helpful in my journey.

Q: Isn't kube-apiserver and kube-proxy the same? I see both Pods running in your cluster?

A: Not the same, but I can see the confusion. Both kube-apiserver and kube-proxy are are essential components that make up the Kubernetes control plane. Yes, as you observed, you see them running as Pods in the kube-system namespace with `kubectl get pods -n kube-system`. Kube-apiserver is the access point for administrators to control the state of

Kubernetes. All those CRUD instructions run through the API. Start a Pod, list the Pods, change the Pods, delete the Pods. Kube-proxy is the route your users will access your services and applications. Public traffic enters from the ingress and routes to the kube-proxies running on each node in the cluster. Kube-proxy is servicing the round-robin load balancing to the Pods on the Nodes.

Q: What do you use for testing (unit testing/integration testing) your Kubernetes infrastructure?
A: Use the tools you use today. I used to use TestNG. Other frameworks like JUnit, Cucumber, Selenium, etc will also work. I would suggest also looking into consumer-driven contracts (specifically Pact), as it's part of your obligation to improve your maturity model. I hope to cover consumer-driven contracts in Kubernetes in 2021 for O'Reilly.

Q: What is a service mesh?
A: We will cover this in Part 2 during week 1. It's a fantastic framework to add on top of Kubernetes as it does a better job at Connectivity, Security, Traffic Control, and Observability. When I say - "When (there's some kind of trouble), I am not slow. It's hip, hip, hip, and away I go!", you'll soon understand why.
https://istio.io/latest/docs/concepts/what-is-istio/#what-is-a-service-mesh

Q: If there is a kernel panic, will it affect all the containers/Pods on the hosting Node?
A: Yes. The kernel is running all those containers on the Node. Kubernetes embraces these failures as part of its resiliency architecture and you should too. In fact, you should be encouraged to run on your cluster a chaotic type of service that periodically injects a kernel panic on purpose. If you are comfortable with this, then this is a metric of your team's high maturity model. Remember the 8 fallacies of distributed computing? Specifically, if you trust the "Topology doesn't change", then your customer will someday suffer. Your nodes should not be treated as snowflakes. They should be readily replicable. Fortunately, the reconciliation loops in the Kubernetes controllers and schedulers will ensure all those lost Pods are started somewhere else on your cluster. The Node that failed should be drained and removed from the cluster as part of the automated Node scaling (cluster elasticity) that needs to also be in place. A healthy observability system would pick up on all these failure and reconciliation events. If you get a full night sleep during this chaos, that's also a good measure of your maturity model.

Q: I'm assuming round-robin is the default algorithm for load balancing?

A: Yes. The services can route traffic-based iptables (default) or IPVS (newer, faster). With iptables its round-robin, with IPVS there are many more balancing choices that an administrator can tweak. Often the balancing choice may not always be the biggest solution when it comes to performance concerns. More about the balancing options are here.

Containers

Q: When public containers are referenced as image names and versions in YAMLs or imperatively using kubectl, are they pulled from Docker Hub or somewhere else?

A: Typically yes. However, since November 2020, the public and anonymous pulls from Docker Hub are now rate-limited. Different clusters can be set up to pull from different public registries and mirrors. It's really up to the configuration of each container runtime engine installed onto each Node in the cluster. Kubernetes does not directly manage the lifecycle of containers. Instructions go out to each Kubelet to manage the life cycles of the Pods. The Kubelets are even agnostic from the containers as they delegate to the container runtime interface (CRI) which connects to the container runtime engine which is the component responsible for the container images. Typically they point to Docker Hub, but really most images come from the Google public registry mirror. It depends on the defaults of the specific type of Kubernetes cluster you run. You can also reference images from local and private clusters. Here is a scenario that shows you how to set up and pull images from a private registry.

For any company serious about using Kubernetes, you should set up a registry where all containers are privately pushed and public images are cached. This private registry holds container images that are scanned for license compliance and security threats. A private registry can be purchased as a service from your cloud vendor or run on a Kubernetes cluster such as Nexus, Artifactory, Harbor, and Quay to and here are a few more.

Q: You mentioned that containers are packaged microservices. What about shared components in the packaging that different microservices/containers might share?

A: At runtime, dependent components within each container are complexly isolated and no sharing at runtime can or should be encouraged across containers. For instance, if you have 10 different microservices written in Spring Boot, each container gets its own JRE, and they do not share JREs across containers. While you may have a problem with this replication, the isolation of containers and each of their contents far outweighs any notion of sharing these runtime dependencies. Remember each container is running on a variety of hardware spread across your cluster's datacenter. Each container needs to have high cohesion and low coupling to ensure its security and atomicity. However, if your concern is ensuring the use of common dependency versions, then setting up a hierarchy of base containers and building off common base versions and dependencies can be achieved at build time.

Q: The container image in this example is nearly 80Mb for just a simple function which can be done in C and it will be less than 1Kb. Can we place any C or C++ programs in containers?

A: You package your compiled or interpreted applications in containers. There are often many other items in a container, besides just your application. For instance a Java application jar file would be packaged into a container with the JRE. Your compiled program binary would still be <1Kb inside the container. The container can hold other items such as OS utilities or other dependent applications that your application may reference or depend on. If you just have a small compiled application you can package it in a container what declares just FROM scratch. Where scratch literally means no other base artifacts and is just a placeholder for a nil base container. You place you binary application in the scratch based image, and it will be that small. However, most people package applications in at least distroless or Alpine based containers.

Q: Is it similar if you do something like dockerImageName:int or dockerImageName:dev in our private repos?

A: No, don't do this antipattern. …

Q: Why get too crazy about container image sizes? Will we save money for all this trouble?

A: Like anything, with containers there is a general rule of diminishing returns. You are correct, that it's important not to get too pedantic to make the perfect container. The container image size is only 1 of 8 factors that are part of the Distillation pattern that I outlined. All 8 points are important to work on, but you must be a practical engineer and do what is best given your time and resources available. However, I want to emphasize that most people generally do not do enough with optimizing containers, at least addressing some of the 8 distillation rules. For instance, any container analysis tool will report that more than 50% of your applications running in containers are running as the root user. This will not fly with your security regulations and will have to be addressed (e.g. switch to use distroless containers). If you don't bother to remove the compilers, debuggers and other development tools from your containers, that's also a hard conversation with SREs and security people when things go wrong. If you get into good practices early and follow ideas like automated governance, then you can address that majority of the 8 factors in the distillation pattern.

Q: Do multiple layers in a container add latency and the number layers that need to be managed?

A: It's really the size of the layers that contribute to the size of the container image. Physically this is the size of the tarball that has to be transmitted over the network wires. The more artifacts you have in your container, the more that has to be transmitted. The number of layers does not always indicate larger size, however each layer has a size and each layer added can add to the size. It's really the size of each layer that is the bulk of the image size. Now, each layer is stored with an associated checksum. When a layer checksum matches an existing layer that is already cached, then the container engine gets smart and avoids downloading layers that it may have already cached. This is a huge saver especially when a container image is updated and only one layer is modified. So have lots of small layers where there are common things you do across different containers, you can greatly benefit from this cache. For instance if the majority of your containers all have the command "FROM gcr.io/distroless/python3" for all your Python microservices, you can see great sving in transmission costs and latency in starting these containers. The order of layers and what you put in each layer has to be managed for this efficiency. I would recommend a Dockerfile linter to help you optimize these layers. To get better at drafting in the DSL of Dockerfiles, seek the teachings from Raju Ghandi.

Pod Objects

Q: How are Pods in Kubernetes different from containers?

Kubernetes manages the lifecycle of Pods. Each Pod is a grouping of one or more containers. Typically you are encouraged to have a single Pod for each container, but there are very good architectural reasons to have more than one container in a Pod. When Kubernetes creates a Pod it is assigned a virtual IP and a small and isolated file system. Because each Pod in a cluster has its own IP, the container in each Pod never has port conflicts with all the other applications in the cluster. From an IP endpoint perspective, each Pod appears to be a separate server machine even though you will typically have several Pods running on each node in the cluster. One strength of Kubernetes is it manages all the virtual IPs of all the Pods and Servers along with a DNS so you don't need to worry about all those changing IP addresses.

A Pod is assigned a virtual IP where one or more containers can be grouped with that virtual IP. The Pod is also assigned a virtual file system. The actual files are on the host Node where the container processes are running. When containers are in the same pod, this means they are actually Linux processes on the same host Node. Because they are on the same host, containers in the same pod can send files, messages and connect to each other via localhost very efficiently. More on this next week.

An application such as a script, a database, a utility, or a microservice are packaged into container images. A container runtime engine can read these container images and run the application defined in the container, along with its dependencies as a completely isolated process on a host machine. The host machine can run numerous containers on the same hard,

where each container is isolated from the other containers. Isolated from the sense of the file system, the user and hardened security boundaries, even though they are all running on the same host machines.

A Pod is a collection of one or more containers on Kubernetes. When Kubernetes creates a Pod it allocates a virtual IP endpoint for access to the Pod and a virtual file system that is shared among the group of containers in the Pod. Typically you group one container in a Pod, but there are designs that work nicely with multiple containers in a Pod. Kubernetes, though the kubelet just manages the life cycles of Pods. If a container in a Pod is deemed unstable or unresponsive, the entire Pod is restarted. When an application needs to scale, Kubernetes will scale the Pods, not the containers within a Pod. Pods are the atomic unit of management for Kubernetes.

Q: My organization is going to use OCI images instead of Docker. In terms of deployment, nothing changes, right?

A: This is correct. The deployment is the same. This is the correct move to use OCI images. The YAML manifests don't care about the container image format. Your K8s cluster just needs an engine that runs OCI containers, which is standard. In fact, the tools that build OCI container images can read your existing Dockerfiles. There are several tools that build OCI container images, including the docker CLI command. Others include buildah/podman and Kaniko.

Q: Could you elaborate on the differences between Pod, ReplicaSet, and Deployment definitions?

A: We cover this in Week 1 and Week 3 of Part 1. A Pod is simply a declaration of a grouping of one or more containers. When created all the containers in the Pods are started on a single node. A routing table entry is made and a virtual IP is assigned to that Pod. In week 3 we will talk about the advantages and disadvantages of having more than one container in a Pod. The ReplicaSet wraps the notion of a single Pod and allows you to simply declare the number of replicated Pods you want. The scheduler will look at the number and through a bin packing algorithm will determine where throughout the cluster where all those replicated Pods should be running. The Deployment definition wraps the notion of a ReplicaSet and allows you to gracefully rollout, and rollback new versions of a deployment. Let say you roll out 100 microservices (mycompany/buytickets:1.0.**0**) for purchasing concert tickets. The developer writing the microservice fixed a bug in the service and your CI pipeline produced a new version of the container that packages the new microservice version, mycompany/buytickets:1.0.**1**. If you apply the new YAML Deployment declaration, then Kubernetes will gracefully shut down the old Pods container version 1.0.0 and start new Pods, all 100 of them with version 1.0.1. You can try this in the First Application scenario.

Q: If the Pod is not running, how does the Kubelet know whether it is deleted already or needs to be restarted? Will it check with API Server for clarification?

A: The Kubelet is responsible for reporting the status of the Pods it was quested to create. If a Pod fails, the Kubelet is not responsible to start a new one, instead, the reconciliation loop in the Pod controller and the scheduler will figure out the best place on the cluster to restore a Pod. The Kubelet only has visibility to the activity of the node, whereas the scheduler has the visibility of the cluster as a whole. Therefore, it's the scheduler that chooses the best place where Pods should be running or restarted. Also, if one or more containers in a Pod fails, the Kubelet will not start a new container. Instead, the whole Pod is destroyed and, again, it's up to the scheduler to figure out where a new Pod will be constituted.

Q: Can you use namespaces to group related Pods?

A: Yes. That's a common question when people first learn about namespaces. "OK, it's sort of like a file folder or a bucket, now what do I put in that bucket?" The answer is it depends. First, a namespace allows you to associate a logical grouping of Kubernetes resources together. Then you can ask, Kubernetes what is in this namespace? (`kubectl get all -n kube-system`). Namespaces can also have resource limits imposed. Declarations like things in this namespace should not consume more than X CPU and Y memory. Namespaces can be assigned labels and annotations to tell others the purpose, flavor, or describe the contents of a namespace. For instance an attribute of "watch: cpu,memory" might tell an observing system to keep an eye on these metrics. Lastly, network security rules can be applied to a namespace to limit and hide traffic going to or coming from the services in the namespace. With all of this, image namespaces give you a way to logically group isolated objects in your cluster using namespaces.

Q: How do I inspect what a Kubelet is doing with a Pod?

A: There is a Kubelet process on every worker Node in your cluster. Each Kubelet logs to /var/log/kubelet.log on the Node. The same is true for the kube-proxy on every Node. However, for a large cluster, you don't want to spend time hunting down which node to go to get this information. Kubernetes will aggregate events and you can simply ask for the aggregated event list. `kubeclt get events`. There are a myriad of other techniques for debugging a cluster. See Debugging. Try the scenario Kubernetes Observability: Basics.

Q: Is there a way to see the Pod logs from the CLI?

A: Yes, the scenario "Kubernetes Logging" has not been published to O'Reilly yet, but can be run here.

Q: Do the Annotations values appear in deployment logs?

A: No, and other select data such as labels also do not appear. If you run the Katacoda scenario Kubernetes Logging, you will see in step 3 of 5 a deployment of *random-logger* has been created and the deployment has both a label and annotation applied. However, on step 3 the *get event* command does not reveal the labels or annotations. In fact, change the command to "kubectl get event --field-selector=involvedObject.name=random-logger **-o yaml**" and still the extra metadata in the event log reveals only select information in the metadata. However, the event log does have enough information for you to extract the deployment and obtain the annotations with a second query, assuming that deployment is still relevant as the event is reflecting a state in the past.

Q: A rogue program could potentially consume too much memory CPU and memory. Where do you specify resource limits for a Pod?

A: You can apply restrictions on Pods, Namespaces, and Nodes. I will cover this in Week 3. You can also run this scenario, Kubernetes: Define a Pod's Resource Requirements, that describes how to declare limits on Pods.

Q: When you have multiple containers in a Pod, is the startup deterministic (always #1 then #2)?

A: No, and with most things in Kubernetes try not to get lured into that way of thinking or trying to add some mechanism for enforcing order. Even if the containers started in the order they seem to be listed in the manifest file, what determines when the container is ready for service? That logic will vary greatly depending on the application. Instead your application should search for eventual connectivity and consistency. Why trying to connect to a service, you may want to make connections in a retry loop with a timeout. Hope for the best and defensively code for the worst. The initContainers feature is one place where order of execution is guaranteed, but this is one for containers that help a Pod get started, and would not be used for patterns such as sidecars.

Q: Will there be a name conflict if all the containers have the same name Nginx?

A: No, because all the containers are named and isolated in the Pods in a namespace. If you add a Pod named my-pod, then try to do it again in the same namespaces then Kubernetes will tell you the name already exists. You can create the same Pod with the same container in a different namespace and there will be no conflict. If you want to create 100 Pods with the same name with the same container, then a ReplicaSet or a Deployment can be created. Done this way Kubernetes will recognize them as clones as part of the same set.

Q: How can we configure the number of containers running in a Pod?

A: First let's be clear that while you can define replicas for Pods, you cannot define replicas for containers in a Pod. Scaling does not occur within a Pod, just the Pods and sometimes the Nodes. You can have more than one container in a Pod and those containers are typical of different types and purposes. The multiple containers in a Pod work together with a solution within that Pod. This leads to the next question.

Q: In which scenario we put multiple containers in a pod?

A: I cover this in week 2 of Kubernetes Fundamentals in 3 weeks. There are a few named patterns such as Ambassador, Adapter, and Sidecar that rely on 2 containers. There are also proxy containers, initContainers, and ephemeral containers too.

Q: How can we estimate the number of pods on a node?

A: It's really the kube-scheduler that decides where Pods are run. As a developer, you can declare to Kubernetes how much CPU and memory a Pod will consume. Given the size of the Pods and the number of resources on a Node that will determine the number of Pods on a Node. Generally, the larger the worker node and the smaller your Pods, the more Pods per node. The question really is more about how many Nodes you will need in a cluster to effectively run your applications. The number of Nodes has a more measurable impact on cost. Of course, you could have lots of little inexpensive nodes or just a very few large and more extensive nodes. The calculations are not easy. It's best to start with medium-sized nodes for a cluster and start deploying your applications. From there you may want to tweak the number of node and node sizes based on performance feedback. Start with a small group of medium-sized nodes and grow from there.

Q: With multiple containers in a Pod, for any pattern (e.g. Sidecar, Ambassador, Adapter), should I use HTTP, HTTPS, IPC, or gRPC. Isn't IPC the fastest?

A: It depends on your needs. Each protocol has its set of advantages and disadvantages. While IPC may be faster, it might incur more coupling between your containers and limit your ability to reuse the container for other purposes in other Pods. Performance may not always be your primary goal. Perhaps you need regulatory compliance for security and secure sockets may be an easier way to achieve compliance, albeit potentially slower than IPC. While many people prefer REST over HTTP for open simplicity, others have found where there is a large amount of data to transport, then sometimes gRPC is faster, yet implies more coupling with protocol buffers. Perhaps reading and writing files to the mutual "emptyDir" file mount between the containers will be found to be both optimal and may still meet your security requirements. Much of your decision will depend on the nature of the data shared between the containers (size, frequency, security, coupling, ease of development/testing).

Q: Isn't the Adapter pattern similar to the Ambassador?

A: An Ambassador is a *proxy* for the primary app so the main container does not have to be concerned with the multitude of various connections. It isolates the complexity of multiple network connections for providing the needs of primary application. The Adapter is a single connection between the primary application and a single data source. The Adapter provides *transformation* of data between the primary application and another service or data source. More about these patterns is blogged here. These patterns gained notoriety in object oriented (OO) patterns ideas. What other OO patterns can you apply to containers and Pods? Here is a solid book to start that thinking: Design Patterns: Elements of Reusable Object-Oriented Software.

Q: Where would you use Job vs initContainer?

A: The only common ground for thoughts between these two patterns is they run a step or series of steps then complete. Aside from that they are *very* different.

initContainers are responsible to run a series of containers sequentially and successfully before a Pod is considered successfully started. The one or more containers are only scoped to assist inside a Pod to initialize and discarded once the Pod is running.

A Job is a container in a Pod that performs some set of sequential tasks and exists. That Job can perform any activity that was written in the container code and runs in a Pod as a Job. A job starts, runs its program, and the program exits. If the Job is to repeat the Job is triggered again and can also be wrapped in a CronJob to have it run periodically.

Q: In the Deployment the labels appear twice (once in metadata and then in template metadata) which ones actually get used?

A: Each label appearing in the manifest in different places has a specific purpose accounting to the manifest's schema. Often the labels are the same to make it easier to read or they are the same because they need to match. Specifically, in the Deployment the matchLabels for the replica selector need to match the metadata labels in the Pod template. These labels are independent of the Deployments metadata labels but are often the same.

Q: When doing the rollout I don't see a change in the status of the old pods, only versions of the newly created pods. Are the older versions of the pods still available?

A: No the Pods are terminated and removed and their internal containers are killed. All ephemeral data is lost. The logs for the pods are still available on the file system of the host Node and can be accessed using the 'kubectl logs --previous' command. However, using a centralized logging solution is best for longer-term access to logs.

ConfigMaps and Secrets for Pods

Q: When you deploy an application, do we first deploy ConfigMap and then deploy the application related Pods as part of the same service/deployment?

A: Kubernetes can also be called a state machine. Good state machines should be tolerant of instructions arriving asynchronously and out of order. Part of the nature of a declarative system is not having you worry about the order of execution, as that starts to hinder you with imperative thoughts. "Just tell me what you want, I'll take care of the execution details." If you start a Pod and it references a ConfigMap or a Secret and those objects do not exist then the Pod will enter the state "CreateContainerConfigError". We would expect that. The controller for the Pod will periodically retry to start the Pod. If you then declare the ConfigMap that the Pod references, then the next time the reconciliation loop comes around the state will be corrected. I would not worry too much about the order of execution and let the controllers figure it out.

You can try this experiment for yourself. Go right to step 4 of 7 in the scenario Kubernetes Fundamentals: ConfigMaps and Secrets. Apply the consume-via-env.yaml. This will start a Pod that references a ConfigMap, but because you skipped right to step 4, the ConfigMap is missing. Inspect the Pod with `kubectl get pods` and you will see the state in error. Go back to step 2 of 7 and apply the ConfigMap YAML. Type `watch kubectl get pods` and in a few moments the Pod will move from "CreateContainerConfigError" to "Completed".

Q: For secrets, how can certificates and service accounts be passed down to Pods?

A: Secrets are defined in Secret manifests with "kind: Secret". These secrets are mapped to the containers as either environment variables or read-only file mounts. The mapping is defined in the Pod manifest. An example is in this scenario, Kubernetes Fundamentals: ConfigMaps and Secrets.

Q: How do you make sure that a Pod restarts when a ConfigMap or Secret changes?

A: There are a few techniques, all are not as seamless as we would like.

- One technique is you connect a ConfigMap or Secret (configuration values) to an application in a container using a file mount. In the declarations for a Pod you can declare a read only file mount where Kubernetes will map the configuration values.

When the data is updated, the values will be updated at that file mount. All you need to do in the application is always read that file each time you want the current value(s). One downside of this technique is it can take up to 5 minutes for the Kubernetes controller to update the value in the file.

- Another technique is manually run “kubectl rollout restart deploy/{deploymentname}” each time a ConfigMap that is referenced by the deployment. This will shutdown the old Pods and start new Pods that will read the new configuration values. It’s a very manual approach.

- Another technique is people will create a sha of the ConfigMap and Secret yaml files. This sha will be inserted as an annotation value in the deployment yaml. This can be done using coded logic in a pipeline. When a deployment is applied to a cluster, using “kubectl apply”, not “create”, then Kubernetes will see it’s a new deployment and will perform a rolling update of the Pods.

- Both Helm charts and Kustomize in Kubectl have the ability to detect associated Secret and ConfigMap changes and can perform the rolling updates when their upgrade features are requested.

- This is a controlling service that you can run on the cluster called “Reloader”.

All of these different techniques (hacks) at the tool level are indications it’s a feature missing at the Kubernetes controller level. https://github.com/kubernetes/kubernetes/issues/22368 and https://github.com/kubernetes/enhancements/pull/948

Q: For environment variables that are Secrets like passwords, how would you pass those in? Would Hashicorp’s Vault work for that?

A: See the next question.

Q: I've read that it is not recommended to populate environment variables with Secrets as they might show in describe output or in logs or standard out. The advice was to always mount them as files if possible. Is it something to be concerned about?

A: Yes, there is a possibility that secrets can be revealed and attacked in this manner or other techniques. Secrets needs to get to your code somehow and should not be baked into the application to follow the guidelines of the 12 factor applications. Kubernetes Secrets can be provided to your container from an environment variable, a command line argument or as a read

only file mount. In these cases you still have the risk of the information getting to the logs or standard out. While these channels are convenient you have to weigh that risk. Mounting secrets to a file might make the most sense with these choices. However, you could add a secret management system such as Hashicorp's Vault as a service to Kubernetes. They have a options for higher levels of secret protection, but you need to weigh those techniques with risk over convenience. Explore the Katacoda scenario Injecting Secrets into Kubernetes Pods via Vault Helm Sidecar written by Hashcorp. I'm fond of this sidecar solution.

Probes on Pods

Q: What do you do if an initial health check is unhealthy? How does the Pod convey that it is ready?

A: Containers in a Pod have a life cycle. The Kubelet that talks to the CRI interface that talks to the container runtime engine can get the general health of a container. If you do not define a liveness definition for your container, Kubernetes will assume the health of the Pod is the health of all the containers in the Pod as reported by the Kubelet. If you do define a liveness probe your application can report if it's healthy or not. All you need to do as a software developer is report, "I'm alive!", e.g. HTTP status 200. If Kubernetes does not get that response in the allotted time you specified, then your Pod will be killed and restarted. To answer your question there is nothing you need to do if your health check is unhealthy. If you find a Pod is constantly restarting, it may mean your liveness probe is too fast and aggressive. You may also just have a bug in a container in the Pod that is preventing it from becoming healthy. There is a new Probe in Kubernetes 1.17 called the Startup probe. This is another place where you can tell Kubernetes, "Hey I'm still starting up, so I'm not alive or ready yet. Please hold your horses before you decide to kill my Pod". I will be writing a new Katacoda scenario on the Startup probe, but in the meantime, you can experiment with the Liveness and Readiness probes in this scenario.

Q: Probes scenario How did they crash the container by executing /unhealthy API. What's written in that API? How did they crash the container by executing /unhealthy command? What's written in that API, which would affect the entire container.

A: The source code is here, there you will find an isHealthy boolean. To exercise this testing feature, all the API has to do is set this flag in the code and the next time Kubernetes pings the liveness probe URL, the service simply returns the HTTP status code 500, Internal Server Error.

Keep in mind that when writing your application, do not rely on this probe url to force your application to restart. If you want your container to exit, then simply terminate your application normally with or without an error code. If you relied on the probe, then the logic in your application would be tightly coupled to the notion that it's running in a container in Kubernetes. Your applications should not be coded to be aware of containerization nor Kubernetes.

Q: In the Probes scenario, in Step 3 under title Crash Service, it seems to say that an unhealthy endpoint returns HTTP 500, but that call returns 200.

A: The wording in the scenario needs to be clarified. The REST call you are invoking is setting a mode in the application to cause the liveliness problem url to return 500, this REST call is successful with 200. The point is each time to invoke /unhealthy then the RESTARTS status for the Pods will bump by 1. It takes a few moments between the /unhealthy request to have the liveness probe file which in turn will restart the Pod.

I will be creating a new Probe scenario that includes the new Startup probe and will make this instructions clearer.

Q: Can probes be specified at deployment.yaml level?

A: No, probes are defined at the container level. Remember that there is potentially more than one container in a Pod, and therefore constructs like Deployment, DaemonSet, Jobs, and ReplicaSet that all reference Pods will also rely on the probes to be defined at the deepest level where the containers are defined.

Q: In the Probes scenario, what is the meaning of this command: jsonpath={.items..metadata.name}?

A: Kubectl can read large amounts of configuration and state information from the cluster. This data can be obtained in various formats such as text, yaml and json. A switch like -o json will tell kubectl to format the response data in a json stream. The command jsonpath is a query technique to dive into the json stream to extract a subset of information.

Q: In the Probes scenario, it references the same endpoint for the readiness and liveness probes. My understanding is this is not recommended and considered a bad practice as they are different things and serve different purposes. Is that correct?

A: It dependents on the nature of your application and the types of probes. Generally they do mean different things, and sometimes keeping them as different requests will keep things neater. If you are using the very common HTTP status return from a HTTP URL, then often the status is enough to provide information to the outside world about your application state and health. Your application should not be aware that it's running in a container, not should it have logic be aware of Kubernetes. If you provide endpoints in your application for liveness, readiness and startup probes, then Kubernetes concepts will start to bleed into your application. A HTTP status of 200, 102, 500, 503 may offer enough discriminating information about state

and health with sufficient decoupling. I would generally ask, how would you report the state of any webservice outside the confines of Kubernetes? This is why ideas like HTTP status and the wide ideas of the Semantic Web are good ideas to follow to keep your applications decoupled from the implementation of the mesh they run upon.

Q: How can I view the entire JSON using the Kubectl command?

A: kubectl has the -o switch to return the query data in various formats. Try this for json, “kubectl get pods --selector="name=frontend" -o json | jq .”

Q: In the Probes scenario, in step 3, what is this pod= line doing?

A: When a deployment is created and Pods are started, then each Pod name will be assigned a unique name for identification. This name is helpful to know to perform further queries on a specific problem, such as getting logs or wanting to execute a specific command inside a specific Pods. These unique Pod names can be discovered at runtime. In this scenario there is one Pod with a selector called “frontend”, but we don’t know the actual assigned full name of the Pod that was instantiated. This command extracts the unique Pod name identifier for the Pod that matches the attribute “frontend”.

```
pod=$(kubectl get pods --selector=""name=frontend"" --output=jsonpath={.items..metadata.name})"
```

Q: FYI 1000m cpu = 1.0 CPUs, so 1024 makes no sense as a CPU measurement.

A: The “m” stands for CPU millicores. A value of “1” with no units, is equivalent to “1000m” which means one core from the CPU on the Node where the container will run. This value can also be a fraction of a core like “0.25” or equivalently “250m”. However, a container is not limited to a single core, so you can specify fractional units above a core like “1.5” and “1500m” or even 1024. Granted 1024 is a magical number usually applied to memory requirements for kilobyte blocks, but it is a valid number for the millicores request. If the millicores are so high the Kubernetes scheduler cannot find a place to run your Pod on a Node, then the Pod will remain in the Pending state until the resource requests can be satisfied. The scheduler events may also produce a message like “FailedScheduling Failed for reason PodExceedsFreeCPU and possibly others”. Also, remember multiple containers can run in a Pod so all the CPU millicore requests for all the containers are added up to represent the CPU requirements for the whole Pod. The scheduler will find a Node where the Pod will fit, even if it requests multiple cores. Linters like *kubeval*, *kube-lint*, and *kubetest* can help you ensure the values your requests are not out of range.

Q: How can you set resource limits in Deployments?

A: Not directly on Deployments. Instead the CPU and memory declarations are done at the level of each container in a Pod. After all it's the container that is requesting resources. These CPU and memory requests are added for all the containers in a Pod, to constitute the resource requests for a Pod. This in turn translates to the grand total of what a deployment needs for all it's replicas. The CPU and memory requests can multiply quickly with multi container Pods with multiple replicas. Without a good understanding of resource needs, your cloud costs can exceed your budgets. New administrators are often surprised when a cluster starts to fail when the user requests increase. Often these teams later discover that CPU and memory declarations have a direct impact on stability. The bottom line is know and define the CPU and memory needs of each container, and do not rely on defaults of best fit.

Other Kubernetes Objects

Q: In the YAML files, in the apiVersion field, sometimes we declare apps/v1 and sometimes just v1. What is the difference?

A: Each Kubernetes manifest in YAML form has a required apiVersion field that tightly corresponds to the kind field. Each manifest declares a resource, a "kind" of resource. The apiVersion applies to the specific schema for the resource and the controller version that knows how to interpret the declarations in that version of the schema. The versions have a specific enumeration defined by Kubernetes. The schema for the actual apiVersion string is found here. I find it a disheartening antipattern that they found it necessary to create their own versioning schema instead of following Semver.org. The string starts with a “v” and can have a number with “alpha” and “beta” in it as well. The string can also be prefixed with a context string with a slash such as apps/. So what version applies to each resource? Based on your Kubernetes version you will have to look at the documentation a kubernetes.io for each resource, but you can also check kubectl api-versions and kubectl api-resources. This scenario may help make this clearer.

The resource versions change and deprecate over time because Kubernetes is continuing to evolve and this eases the migration as these resource implementations improve.

Q: Will we learn about data-persistency in Kubernetes?

A: Yes, Volumes, and StatefulSets in week 3. More in the following answer.

Q: Could you explain Kubernetes storage?

A: Let's work from the inside of your application outward to the persistent store. There are a few layers of indirection that protect you and Kubernetes from being tightly coupled to storage implementations. First, your application can read and write for any file within its file system purview. One of the directories might be a mounted Linux path. Linux file mounts are key for your application to connect to a drive outside the container. In the Pod definition, you define the path between the container file mount path and the PersistentVolumeClaim (PVC). The PVC is bound to a specific PersistentVolume (PV). The PV the representation of the mount to the actual persistent storage. You can sort of think of the PV as the "device driver" for external volumes. There is a wide array of volume types to connect to, and fortunately many of the volume providers banded together to make the Contain Storage Interface (CSI) standard for exposing arbitrary block and file storage systems to containerized workloads. The gory documentation on Volume is here. Try the scenario Kubernetes: Define & Mount a Persistent Volume.

Q: Can you please differentiate between NodePort, ClusterIP, and container IP, ingress?

A: You are asking about the 4 types of Service types in Kubernetes. A "kind: service" is the definition of an object that provides both service discovery and load balancing to all the replicated Pods connected to a Service. This information is declared in the service manifest YAML. The Service can also be of 4 different types: LoadBalancer, ClusterIP, NodePort, and ExternalName. There is no "container ip" or ingress mode for a Service. The 4 types are defined here, but I can also translate. **LoadBalancer**: This service will be for outside traffic coming from your ingress, ingress controller, and your load balancer. **ClusterIP**: This service will accept traffic coming from other calls within the cluster only. **NodePort**: This service will accept calls coming from an exposed static IP on the Node (helpful for experimenting). **ExternalName**: This allows you to connect an external URL rather than routing the traffic to a Pod. Callers will use the Kubernetes DNS to find the service, which forwards it to another URL.

Q: Does it matter if all Nodeport values are the same in a manifest?

A: The NodePort value is 30000 and above. NodePort numbers are global to the cluster scope, so port conflicts across multiple NodePort service types can happen. When you submit a Service of the type NodePort without a nodePort port value then Kubernetes will find an unclaimed port number. If you don't care about the value, this is safe to do. If you try to control what the value will be, you will get into trouble because there could be others that also attempt to claim a specific port number. It's best to avoid this type of conflict and let Kubernetes find an appropriate value. In the Katacoda scenarios I set the nodePort to a specific value so subsequent commands can be fixed to port value examples such as curl commands. In production systems it's best to define and ingress to the services by URLs to service names instead of relying on port numbers.

Q: How does ingress Kubernetes object interact with my load balancer that has its own rules for routing requests?

A: The Ingress object is just a declaration of rules. On your cluster you will need an ingress controller/proxy to provide that actual connection and interaction with your load balancer. There is a list of common ingress controller/proxies for Kubernetes found here. One or more of these controllers should have the implementation details to coordinate with your load balancer and its configurations.

Q: What's the difference between Ingress and NodePort?

A: While both of these address the needs for routing traffic coming into the cluster, they are different. The Ingress is an Kubernetes object that maintains the list of rules of how external URLs are converted to call to services within your cluster. These ingress rules are referenced by the ingress proxy/controller that is associated with your load balancer as described in the answer above. The services in front of your pod that accept this ingress traffic needs to be the service types called “loadbalancer”. If your Pod has a service of the type NodePort then this means traffic will come not through a load balancer, but instead through a public port opened on each Node in your cluster. The load balancer types is typically used for production type of real traffic, where NodePort is used to testing, prototyping and demonstration purposes.

Q: What about egress and managing outgoing traffic? How does this work in Kubernetes?

A: By default the ingress and egress for service within you cluster are all unrestricted You can apply a NetworkPolicy to one or more namespaces to refine the ingress and egress restrictions for service associated with a namespace. The ingress policy in the NetworkPolicy should not be confused with the Ingress object that defines the routing rules for your ingress proxy and cluster load balancers. Adding a meshing solution like Istio to Kubernetes, also provides better opportunities for controlling load balancer ingress, ingress between services in the cluster, and egress from Pods.

Q: Can we access pods located in other data centers?

A: Yes. To keep terms clear let’s assume that each data center means the cluster of nodes that make up a single Kubernetes cluster. Many companies manage multiple clusters. Some clusters are separated for business segregations, to keep customers tenants isolated, to keep testing and production clusters isolated or to host a cluster across geographic global zones. There are many reasons to have multiple clusters. Often your same application containers will be replicated on other clusters. Sometimes there are unique services on specific clusters and you may need to reach out to other services.

In the easiest cases, if the remote service is publicly exposed as a public URL, then the access is easy given the appropriate SSL and latency concerns to geographically remote services.

For intra cluster communication the URL that the CoreDNS service recognizes is in this form:

```
service-name.namespace-name.svc.cluster.local
```

Where `service-name` is the name of your service and `namespace-name` is the name of the namespace where the service was created. Notice the `svc.cluster.local`, this indicates that the URL is `local`, or intra to the cluster. There is a way to configure the CoreDNS to recognize other clusters that are beyond "local". This gets into the ideas of Federated clusters.

Networking meshes, such as Istio while working well for intra cluster communication, also help you tie your applications to other services outside the immediate cluster to other legacy services or other Kubernetes clusters.

Another option is to follow the ideas of Kubernetes Federated clusters, now in its second form yet in alpha, trying to approach beta status. More is here on the Kubernetes Cluster Federation idea that is part of the SIG Multicluster group. There is hope here for a generic cloud native solution for this, but it's been in the works for a while.

Your cloud vendor may offer proprietary cross cluster communication solutions as well. For instance, Google GKE service offers a multi-cluster services (MCS)

This is an area in the Kubernetes upcoming releases that will improve.

Q: I have seen labels mentioned in the metadata and spec sections in YAML manifests. What are labels used for?

A: Any kind of information can be added as labels or annotations. As the number of objects grows in your cluster these become very important. Labels are also used for connecting and associating objects. Labels can also be used to search and query objects based on matching labels. Annotations are like labels, except they are used for identification or annotating, rather than for filtering. This is all further explained in Kubernetes: Define & Query Labels & Annotations scenario.

Q: When Nodes are drained, how does Kubernetes know that the Node has finished draining?

A: Specifically, the Kubelet on every Node reports the state of all Pods on it's Node. Draining a Node involves calling "kubectl drain <node name>". This is a blocking call and will return once all the Pods on the Node have been stopped. The Kubelet on the Node will stop all of its Pods and containers within and as always reports its status.. The Kubernetes scheduler will not ask the Kubelet on the draining/drained Node to start any new Pods. Keep in mind that although those Pods have been stopped, their replacements have started on other Nodes in the cluster. Except, for Pods as part of a DaemonSet. The marked Node is considered drained once all the Pods have been released from the Node. At this point the Node can be safely removed from the

cluster. During the draining process the PodDistruptionBudget is also respected and if there are not enough resources to meet the budget requirements this can block the draining request until enough resources have been made available or the budget requirements are lowered. More details here.

Q: Is it recommended to rely on the Kubernetes objects Namespace and NetworkPolicy to define which service talks to which or to implement the authorisation at service layer via JWT or similar tech?

A: Kubernetes Namespace and NetworkPolicies are limited in what they can do when it comes to your application data plane. The NetworkPolicies are enforced near layer 5 where JWT is closer to the application data plane at or near layer 7. So it depends where you want to define this control and the needed granularity of control. If you control access in JWT or at any layer close to 7, then you may be inclined to add that logic and control inside your app code. This creates frictions against highly cohesive apps and containers. Why would the app delivered to your customers, also include connectivity rules? This also rubs against the distillation pattern for containers. Each time you change a JWT rule do you have to roll out a new version of your container? Perhaps JWT can be done with sidecars. A more appropriate viewpoint would be to compare JWT with meshing solutions that also work directly at layer 7, yet are decoupled from your application services. Explore JWT with Istio.

Volumes and Datastores

Q: How can I initialize a database before an entire deployment can start?

A: This question has several layers to unpack. This answer can be greatly expanded upon, but let's address a few assumptions and scenarios.

First, your deployments that rely on a datasource should have protections in place to assume that the database will not be immediately available during an asynchronous startup, and will fail to function during it's lifespace. Assuming evenal startup, recovery and consistency is a better way to think about connecting to distributed services. Startup order is not guaranteed and creating a scheme to instill order of execution on an asynchronous network creates difficulties. Embracing asynchronicity, startup latencies, and network hiccups is in conjunction with cloud native computing.

Databases can live in different places and forms based on your needs. They can be an external service to the cluster, a database in a Pod with an external file mount to a persistent store, or a full database in a Pod with a more ephemeral data store. A database can be small and dedicated to a single microserice, or be monolithic and available to several deployments on the cluster. These data store flavors create more topics for discussion from the original question. Many situations are available for you as an architect working with Kubernetes.

Lots of databases can be run in a container and mounted to external file stores. In production never put large enterprise databases in containers, like Oracle as it can be a license violation and may not be your best performance option. It's perfectly acceptable to put databases like Oracle in a container, but you may want to limit that to testing and prototypes. A proxying Pod can connect to a database, but it may incur and unacceptable latency. If your datastore is accessed via a Pod or an application in a Pod acting as a proxy, then you can do a few things.

> 1) If you can completely initialize, seed and test a database all at container build time, this is the best option. When the database container starts, it's ready, not work to be done. A lot less to potentially fail.
>
> 2) initContainers run in on the database Pod can ensure your database is not ready before all the initialization completes. This all can be done without probes.
>
> 3) By placing probes on the Pod then the state of your database can be determined with the Pod probes. The new startup probe is helpful for knowing when the database is initialized and the periodic readiness problem can alert you to hiccups. The liveness probe can be helpful for recovering from more catastrophic faults, especially if the Pod is a proxy to an external service, or is a database connected to a persistent, external file mount. By relying on probes then you open more doors to how the database will be

initialized when started, for instance a Kubernetes job triggered by some event. However, the farther you go this way it gets uglier and I encourage instead to init the database at build time when the container is created.

If your database is an external service then there is a Type of the Kubernete object Service where you can define an “ExternalName”. With this you tell Kubneretes an external address that is resolved when the internal service lookup DNS resolves your service name. There is also a “kind: Endpoints" that can be linked to a “kind: Service". However, probes are only available for Pods and not these external services mechanisms. With an external service as a database, then the database can be initialized externally, or with a Kubernetes Job trigger by some startup event, perhaps using something like LiquiBase.

Q: In the scenario Run Stateful Services on Kubernetes I missed how the PersistentVolumeClaims are bound to the PersistentVolumes. I didn't see where they were linked with a name?

A: When a PersistentVolumeClaim is declared, the Kubernetes control plane looks for a PersistentVolume that satisfies the claim's requirements. If the control plane finds a suitable PersistentVolume with the same StorageClass, it binds the claim to the volume. The design intentionally tries to prevent an explicit connection between a PV and PVC. Once a claim is made, nothing else can bind to that volume. Multiple Pods can connect to a single claim with a matching label. More about PV/PVC binding is here.

Q: A PersistentVolumeClaim binds to a Volume. How can you explicitly bind a Volume to a PersistentVolumeClaim?

A: You don’t explicitly couple the two declarations together, instead the Kubernetes controller finds the best match for you. See the answer above.

Q: Many applications are I/O bound; how do you observe and scale when that happens? Especially if the I/O latency is outside the cluster such as a NFS service or an RDS instance. Just adding more Pods won't help.

A: I/O bottlenecks are created by single points as you describe. You can create more microservices across many Pods to distribute the processing and memory load, but if they are all accessing the same persistence then the I/O bottlenecks rear their ugly head. A solution is data partitioning. A place to start is the Service Based Architectural style covered Chapter 13 in Fundamentals of Software Architecture. Sam Newman covers in chapter 4 of Monolith to Microservices the topic of Decomposing the Database. For distributed systems, data partitioning and sharding is helpful, however it can complicate things. But before you start proposing crazy, disruptive new architectures, gather your facts first. I would start with some performance

observability tools tuned to look inside your data stores from your database vendors. There are lots of observability tools for Kubernetes, but in this case the bottleneck is outside the cluster at the I/O point with the storage service, so start observing there first. If you can't squeeze any more water out of the I/O rocks, you should have the data to start floating the ideas of database partitioning, sharding and new ways to change your architecture to add more data I/O pipes.

Q: What are the best practices for ensuring data consistency between pods running in different availability zones?

A: Kubernetes supports the notion of volume affinity when working with multi zones. You can tap into this feature if your application relies on PersistentVolumes and claims. The document on Running in Multiple Zones covers this persistence with zone affinity.

Q: What is Container Storage Interface (CSI)?

A: Before CSI there were volume plugins into Kubernetes, yet were difficult because of the inconsistency in standards, from volumes which all have essentially the same core features. When you make a PersistentVolumeClaim you can associate the claim with a storage class. There is an independent declaration called a StorageClass where you can define the CSI plugin for the volume. More details can be found here.

Q: Can storage, with or without persistence, be provided by services within the cluster?

A: Yes, and I would encourage you to consider doing this more often. After all Kubenretes is a platform where you can define using infrastructure-as-code a whole stack of solutions for your application. That stack often includes storage solutions. Redis is a popular solution to put in a container and Pod, but there are several hundred other data storage solutions ranging from flat files, registries, relational databases, nosql, graph and distributed storage. Running these all on Kubernetes means you are leveraging the ecosystem of scaling, fault tolerance, observability, a mesh and all the other things you expect from Kubernetes. However, sometimes this requirement becomes more work as the data storage requirements span more business features, and grow in size and importance. In these cases you may want to buy a storage service from your cloud provider where they can take care of backup, restore, fault tolerance, auditing, updates and all the other aspects of a high available production service. If you do run a data store in a container, then the backing persistence storage can be in the container, in the Pods, on the Node, on another service in the cluster, or outside the cluster. With all these choices you have to weigh the various levels of safety and ephemeral states. For integration testing scenarios running a relational database with ephemeral storage is a handy solution. There are many volume store solutions that work with Kubenernetes. You can connect to your

cloud provider's storage volumes, there is also CephFS which is a container that offers storage right on Kubernetes. There are many choices.

Q: What options to choose if we use a database as a cloud managed service for initializing schema, tables and data? These are more like pre-tasks that should be executed before application Pod is deployed on the Kubernetes cluster.

A: For services outside of the cluster and for services that depend on external services, it's best to start up, initialize and test these external dependencies before provisioning the cluster. For cloud managed services follow the recommended practices and tools you're comfortable using to establish these services. I tend to gravitate toward Ansible and Terraform scripts. But there are also other cloud native tools like Pulumi that can help. With Kubernetes now in your toolbox you could explore running some of the provisioning and testing workflows in pipelines or Jobs on Kubernetes.

Q: If the volumes are external, does that mean we need to mount a NFS volume? Or even S3?

A: Yes, mounting to external volumes, especially those offered by your cloud provider is a solid choice. Keep in mind there are many choices beside NFS, and many are superior. See volume types choices here.

Q: You showed an example with initContainers with Postgres, thanks for that tip. However, after you spin up the Postgres Pod, can you access the database?

A: Yes. Be sure to place a Service in front of the Pod to access the database on its standard port, 5432. I would recommend a Postgres Helm chart or even better the Postgres Operator to setup a production ready service.

Q: I heard that Oracle Real Application Clusters (RAC) can not be run in containers. It's that true?

A: Yeah, don't hit that screw with a hammer. Oracle RAC is its own thing, unintended for Kubernetes. However, I have seen teams use Oracle Express Edition in a Container for testing services that need to connect to Oracle. I would not use Oracle XE for production, but it's an effective way to get portions of your Oracle based data running on the cluster for testing needs

where the databases are more disposable. There is a reason you will not find publicly endorsed Oracle database Helm charts or Operators. If you are partial to the company Oracle for database solutions and want to run them on Kubernetes, then you might want to consider Bitnami's MySQL Helm chart.

Declarations with YAML and Helm

Q: Where can we specify the namespace in the YAML?

A: In the top metadata section of the Pod YAML manifest. However, if you peg the Pod to a namespace in the YAML file you will lose the ability to deploy the Pods to other namespaces. Many people refrain from specifying the namespace in the YAML and instead pass this and an argument like so: `kubectl apply --namespace testing --file my-deployment.yaml`

Q: Can k8s-compliant YAML files able to be used with other orchestration tools, more specifically Ansible?

A: I have not heard about other systems understanding Kubernetes manifest. There are tools to convert old docker compose files to Kubernetes YAML files. I would not call Ansible an orchestration tool. However, yes there is an Ansible "module" where it will directly recognize the sub-YAMLs as Kubernetes manifests. To be clear Ansible uses YAML files and so does Kubernetes. These are different schemas. The Kubernetes module allows you to place Kubernetes YAMLS into the Kubernetes module section of Ansible YAMLs.

Q: Is there a standardized way of creating Pod YAML files?

A: The fastest way is via the kubectl command by creating an object, specifying dry-run, and outputting to a YAML file.

```
kubectl create deployment my-app --image=nginx --dry-run=true -o yaml > my-app.yaml
kubectl create namespace solution --dry-run=true -o yaml > my-namespace.yaml
```

VSCode also has a nice plugin.

Q: How many kinds of manifest they are and where can I find all the schema/examples?

A: It dependents on what version of Kubernetes you run. Roughly there are about 50. Kubernetes 1.18 has 58. Kubernetes 1.18 has 54 as a few deprecated objects were removed. They are all documented on kuberentes.io in the API section. Study the Kubernetes API scenario too as the different types of objects are revealed in there with this command. `kubectl api-resources`.

Q: How does HELM fit into all this Kubernetes ecosystem?

A: Helm follows the package manager model for the operating system. It's often called the package manager for Kubernetes (if you like to think of Kubernetes as an operating system). Applications or larger solutions are not typically deployed with a single YAML file. Most solutions require a set of YAML manifests with opinionated configurations. This collection of manifests are assembled into a chart. The charts are bundled, named, and versioned. The chart packages are then deployed to registries and installed onto Kubernetes. Typical actions are Helm install, delete, upgrade, and test. Helm also has a templating concept for context environment variable replacement that gets applied based on the type of Kubernetes target you wish to deploy to. Helm is a command-line tool. Avoid version 2 and only use version 3 or later. The charts can be stored in private or public registries. A growing number of common architectural solutions are publicly available on Helm Hub.

Q: If you already have Ansible established in your organisation, would you still recommend Helm instead of Ansible?

A: Hammers for nail, screwdrivers for screws. Both effective tools in my toolbox, just as Ansible and Helm. Use Helm v3 and you may start discovering some features that are different and work better than Ansible. I see Ansible as a wider scope, and use Helm specifically as a package manager for Kubernetes. They are not substitutions, but rather complimentary.

Q: Recommendation for Helm, would you use?

A: Yes, there is also Kustomize that is now built into kubectl, but it only does parameter replacement rather than full package management. I would add Helm to your toolbox and use it when appropriate. The old version 2 of Helm used a security flawed architecture component called Tiller, so it got a bad wrap for a while. Version 3 fixed that and it has evolved. Another tool to consider in conjunction with Helm is Ansible.

Q: How does Bitnami fit into this ecosystem of Helm charts?

A: Bitnami was acquired by VMWare in 2019. VMWare is one of the knights around the Kubernetes round table and therefore a significant contributor. Bitnami focuses on the ease of installing complicated stacks to complicated targets. Helm charts on Kubernetes on a cloud platform would be a solid example. Bitnami continues to use Helm as a tool to deliver opinionated packages to Kubernetes intended with production level settings. Bitnami has contributed quite large and help collection of public Helm charts for us.

Q: What is a CustomResourceDefinition (CRD)?

A: The Kubernetes control plane has a controller manager. The manager organizes all the controllers. Each controller is responsible for specific Objects in Kubernetes. Kubernetes comes with about 50 standard Objects that are all managed by controllers. Objects such as Node, Pod, Deployment, Job, Service, Ingress, Namespace, NetworkPolicy and many more. The interesting thing about the extensible Kubernetes architecture is you can define your own Object types such as Function, Legacy, Workflow, Pipeline, Crypto, PigIron, Puzzle, Riddle, QuantumService, or anything you can dream up. Your new Object types are defined through CustomResourceDefinitions, and yes there is a standard controller for CRDs as well. This topic falls under the Operator pattern that we explore in Part 2.

Q: What is the benefit if we go with an Operator and not installing services directly with a Helm chart?

A: A helm chart will install the application or set of applications that make a solution, but an operator will not only install the application(s) but also run and make changes (manage) the application(s) over time. Helm is a package installer where the Operator is a package manager. An operator can be installed with a Helm chart. When instructed and Operator may run Helm install, update and delete as part of its managing responsibilities.

Q: Do you prefer using Helm charts over single YAML files?

A. Yes. The truth is YAMLs change and grow over time. While you can jam several Kubernetes manifests into a single YAML (separated with '---'), it's not the cleanest, modular way to manage source code. Any change to the deployment means that whole artifact must change. Helm also provides context injection. Not everything in the YAML file is static. Deploying on where you deploy the manifests, some values data will change. Without Helm you have to have some templating or parameter replacement mechanism to tweek these values before deployment. Finally Helm offers a large collection of YAMLs to be groups as a solution set. That set can be named, version and easily stored in registries as a tarball. That's harder to do with just plain YAML source files.

Q. Should we have a Helm private registry in my organization?

A. Yes. If your build pipelines are already producing artifacts such as jars, wars, wheels, install packages, binaries and event container images, then yes. Helm charts are artifacts built from pipelines based on infrastructure as code. The charts are named and versioned (semver) like any other built and promoted artifact. Public charts should be cached and security scanned locally. Chartmuseum is a low key registry for charts. Most common registries such as Nexus,

Artifactory, Quay, Chartmuseum and many other offer registries for Helm charts as well as container images as well as other images. If you have an existing registry see if they offer a chart registry plugin or extension. There is recent activity to ensure Helm charts can be stored as an artifact in a standard container image registry. After all it's just a tar file that's named and versioned. For large enterprises Quay has some nice features. Be sure your registry has security scanners installed as well. So much DevOps, so little time.

Containers

Q: Do I need to be good at Containers before trying out Kubernetes?

A: You can jump right into Kubernetes without needing to know containers. However, eventually, if you get far enough you will need to be good at producing containers since the building blocks of Kubernetes applications rely on containers. Learning how to initially build and run containers is not too onerous.

Q: What's the difference between Docker and Kubernetes?

A: Docker is a company name, it's the name of a command-line tool, and it's a common name for the architectural pattern called "Containers". So let's assume you asked, "what's the difference between Containers and Kubernetes". Containers are a way to run your application as a process on Linux that is so isolated that it appears to be the only application on that machine. There are other definitions too, but basically it's a way to package your applications and associated dependencies into an executable, protected, and isolated process. Kubernetes is a way to orchestrate 1 or thousands of containers on a cluster of machines in a datacenter.

Q: Could you please elaborate a bit on REST, gRPC, GraphQL.. When would you use one and when another? What are the pros and cons?

A: Yes, my explanation was short and there is much to this decision making. This article may nail this answer, APIs REST, GraphQL, or gRPC – Who wins this game?

Q: What are cGroups?

A: cgroups (abbreviated from control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. cGroups are a key kernel feature that define and restrict container behaviors and bounds.

Q: I thought containers do not have an operating system?

A: You are correct, they don't. Your application in the container is leveraging the Linux kernel of the host operating system that is running all your containers on that Node. However, if at the top of your Dockerfile you declare FROM Ubuntu, then inside that container will be an isolated filesystem with all the artifacts packaged with the Ubuntu distro. This just happens to be running

on the kernel of the host operating system, say Debian. Because of this, people feel like inside the container it's an Ubuntu OS? It's a turtle in sheep's clothing.

Q: Container instances are supposed to be state-less. Does that mean any state that an instance may want to save, should be sent to another container instance that runs a database?

A: Don't get hung up on stateless. It's really all about where your file mount is for persisting data. If you store data in the file system in the container and the container goes away, you lose your data. Don't cry, as this is a great idea for temporary caches. When you run your container you can provide it an external volume path and mount to that. When your application in the container writes to the mount, it's to an external volume. When the container goes away the data is still on the volume, because it was an external mount for the container. Thank heaven for Linux file mounts. This works for volumes on the host operating system that runs the container. But in Kubernetes, these host operating systems are on the Nodes, and in Kubernetes, land Nodes are also considered ephemeral as then can either die, drain, update, or simply scale away. If the Node goes away, the volume is also lost. In Kubernetes, there are volumes that can be mounted in other containers or on more persistent stores in the cluster, such as an NFS file mount or EBS. Through the Kubernetes objects PersistentVolumes and PersistentVolumeClaims your container can mount to volumes that have a much longer life than your Pod.

Q: What's the difference between CRI and container runtime engine?

A: Container Runtime Interface (CRI) is the standard definition for the API that can control a container runtime. The kubelet on the Kubernetes control plane only speaks to container runtime engines through the CRI interface. There are many container runtimes available, and a subset of them implements the CRI interfaces. The most notable CRI compliant container runtime engines are CRI-O, cri-ontainerd+containerd, dockershim+containerd, cri-plugin+containerd, frakti. The old CoreOS Rkt engine, and subsequently Rktlet, has been retired as some of those ideas morphed into CRI-O promoted by RedHat. I'm partial to the CRI-O project. https://kubernetes.io/docs/setup/production-environment/container-runtimes/ Don't let the soup of acronyms confuse you as there is also the OCI Runtime Specification. This is the specification for the image format that container runtime must understand to start, run, monitor, and stop containers.

Q: What is a long string that is displayed when running some docker commands in docker Redis?

A: In this case:

```
docker run -d redis:latest
```

da850c58a7175a904a995bdf42ca29e98035055c9bcdd17c3877f640d7b03697

This is the ID of the container instance. Each time you execute the run command a new, unique ID is created. Even if you reference the same Redis container image each time a new Redis instance is started. Each ID correlates to a Linux process ID. If you run docker ps, the same ID will be shown, albeit truncated for printing efficiency. Keep in mind the Image ID (SHA256) is different from the Container ID. Both IDs are SHA256.

Q: Why do we need to yell at the Dockerfile?

A: All the 17 reserved commands for the Dockerfile DSL and all are conventionally written in UPPERCASE. You should follow that established norm. Shouting is a monochrome way for syntax highlighting, albeit a bit rude. At least there's no semi columns, mustaches { }, exception handling and null pointers. If you want to be more polite try Bazel as the language for creating containers. And what's with Dockerfile with an uppercase D but not on the F. No snakes or camels found here.

Q: In Dockerfiles what's the difference between RUN, CMD, and ENTRYPOINT?

A: This is one of the most asked questions among these Dockerfile commands. Unfortunately, CMD and ENTRYPOINT are not self documenting terms. Here are some postulates:

- ENTRYPOINT specifies the command that will always be executed when the container starts.
- CMD specifies the default arguments that will be given to the ENTRYPOINT command.
- If you want to make an image locked to a command you will use ENTRYPOINT ["./myapp"]
- If you want an image for general purpose, leave ENTRYPOINT unspecified and use CMD ["./myapp", “arg1”, ...] as you will be able to override this default command and arguments by supplying arguments when you start the container.
- RUN occurs when the layers are being produced at build time. RUN will execute any commands in a new layer on top of the current image and commit the results.

Once you write a few Dockerfiles these rules will start to make more sense. This blog nails it.

Q: Is it possible to find the Dockerfile inside this tar?

A. No, just as you would not likely find SQL in a database or Java source code in a class file. However, programmatically you can COPY the Dockerfile in as a plain file artifact into a container image layer.

Q: Is there a tool to verify the level of OCI compliance that a build tool generates?

A: Yes: https://github.com/opencontainers/image-tools. I hope to create a scenario for this. I would also encourage you to run a security scanner for containers such as Clair.

Q: Do you create a Dockerfile or docker image to deploy/pull in Kubernetes?

A: A Dockerfile is infrastructure-as-code that defines how a container image builder should build the container image file. Once the file is created, really just a tar file, it is then "pushed to" and "pulled from" (file copied) to a registry. When you declare a Pod on Kubernetes the Kubelet will invoke a CRI call to the container runtime engine to start a container. The Container runtime engine will pull that container image (tar file) from the registry, crack it open and run the contents of the container image. The only thing you store in source control is the Dockerfile text file. The container image should be built by your CI/CD pipelines.

Q: Why is the tar archive (tarball) the chosen format for container images?
It's one of the oldest and most ubiquitous formats for bundling multiple files into a single file. Since a container image is a collection of files and the tools and language APIs for tars are a well beaten path, it makes sense to use tar instead of inventing a new wheel. The image manifest is in the format specified by the OCI specification

Q: Let's say if we get a container image from a vendor, how do I run that through Docker instead of Docker Hub or other public registries?

A: A container image is just a tar file. However, Kubernetes and Docker pull images (tars) from registries. These registries can be either public or private. You will want to publish private docker containers to your private registry. Soon you should consider setting up a container registry for local development. Many cloud vendors offer private registries that you can purchase as a service. You can also run registries locally on Kubernetes. Harbor for instance. Here is an example: https://learning.oreilly.com/scenarios/kubernetes-pipelines-registries/9781492078951/. Most vendors will offer containers to be pulled from a registry. The registries can be public or private and accessed with a key.

Q: What kind of garbage collector is baked into GraalVM Java executable output?

A: There is a special one for GraalVM native images. Here is an explanation: https://blogs.oracle.com/javamagazine/graalvm-native-images-in-containers

Q: Is there a container mechanism for non-*nix systems?

A: Windows servers can run Windows-based containers.

Q: Can .NET frameworks run in containers on Windows?

A: Yes. If you are just using .net core then those can be run in Linux based containers. If you are using .net full then they must be run in Windows-based containers and then are limited to running those on Windows hosted machines. With Kubernetes you can label your pods and nodes so they have "affinity" to run together.

Q: Can one Kubernetes cluster run both Linux and Windows containers?

A: Yes with declarative labeling and node affinity.

Q: Do you believe that Windows Docker Containers share the same characteristics as Linux Containers?

A: Yes, there has been a concerted effort to make the experience similar, however, there are differences in the container runtime engine, but I don't have those details.

Q: Is it possible to run Java projects in container without JVM in a container?

A: Yes, if you use a GraalVM tool called native-image it will compile your Java application to a native binary. An example is in this scenario: https://learning.oreilly.com/scenarios/kubernetes-fundamentals-distilled/9781492078883/ Keep in mind if your application has dynamic class loading and leverage java reflection (such as Spring Boot), then it will not work out-of-the-box as native-image needs to know everything that it will load a compile-time, not runtime. I would suggest projects like Helidon, Quarkus, or Micronaut instead of Spring Boot. I also understand that might be a big ask, but I see the benefits as the number of your distributed services grows.

Q: What is an Alpine?

A: Alpine is a very small Linux based OS. One reason it's small is it uses the musl c library, instead of the more common gnu c library for making kernel calls.. minimal Linux distribution. It's very popular because it keeps your containers smaller than, say Debian based images. However because of the c library difference, extra testing and validation is needed, e.g. with Python and Java.

Q: What is a distroless container?

A. A collection of base containers provided by Google for many of your applications and languages. While not as small as Alpine, they are highly distilled, immutable and secure. Since they are well tested, small and tuned for Google's production needs you will find them good for production. Because you cannot shell into these containers they make it hard to use in development. This is one of the reasons for the recent addition of Ephemeral containers to Kubernetes to ease debugging with Distroless based containers. Generally I use the officially supported distroless images whenever I can. I prefer to stand on the shoulders of the giants.

Q: FROM scratch? What is a scratch container?

A: Yup, it's a thing. It's actually what all containers derive from. The deepest turtle. Sort of like `extends Object` in Java. It's the smallest base container you can ever have because it's actually empty. No files, no assumptions, nada. You can load a binary and execute it. Granted your binary needs to be built statically linked to libraries as there are no libraries in their container to dynamically link to.

Scratch a reserved word for defining a base container. There is no OS, it's a blank container image as a base for all other containers. The lowest turtle you might say for its turtles all the way down, until you get to scratch. No file system, no operating system, no utilities, it's raw. You can load a binary executable into it as long as it's compatible with your host operating system where the container will run. The command "FROM scratch" is a no-op and no layer is created for it.

Q: How about Universal base images (UBI)?. Are they slim and fast to start?

A: UBI images are governed by RedHat. If you're using OpenShift they may be something you want to consider to follow their production level recommendations. The containers are not going to be as small as Alpine, but they have most likely been optimized to balance between security, maintenance, reliability, and minimal size. https://developers.redhat.com/products/rhel/ubi/.

Q: I run COBOL in Containers

A: I stand corrected.
https://developer.ibm.com/technologies/containers/patterns/running-cobol-in-a-cloud-native-way/

Q: DockerSlim vs Dive?

A: Both helpful tools in your toolbox that can help distill your containers. You don't have to make a choice between the two.

Q. How can I increase the quality of the code in each container?

A. Start with static code analyzers near your code development tools. While there are nice plugins for build tools like Gradle and Maven and the like, try to go polyglot with your code analysis at the pipeline continuous integration level. You can run SonarQube with its wealth of language plugins. It proves a nice dashboard and database. You can add it to CI pipelines to break builds. There's plugins for security checks, bug finding and technical debt trends. Automating SonarQube will definitely increase your team's maturity model. Search for the Katacoda scenario "SonarQube" to see how you can run it on Kubernetes.

Q: Do containers share the same host kernel?

A: Yes, containers share the host kernel. A container is just another Linux process on the host OS. The host performs the isolation (e.g. namespaces, chroot, selinux, cgroups etc) to make it look like each application in each container is on its own dedicated host. However, a host can have dozens, even hundreds of containers. (industry average is about 30 containers per host node). You application will eventually make calls to the Linux kernel through musl c or gnu c library calls. Those kernel calls are actually to the host Kernel.

Q: Why is running as root bad in containers and how should I avoid it?

This is one of the biggest best practices to follow with containers, especially with containers targeted for production deployments. By default most containers start as root. Almost 50% of all containers out in the wild run as root. Stop that. Most developers just load their application into a container, and once it runs they move on. Until, one day someone runs a security scan and delivers a surprising report. Do yourself a favor and ensure you get in the habit of taking the steps necessary to change the user and group away from root. To start, just add `USER 9000` to your Dockerfiles and many of those security reports will get thinner. Distroless containers ensure your application does not run as root so deriving from distroless containers you inherit this best practice for free.

Q: There are various guidelines from building container images. Many official images are not inline with recommendation. For example a popular recommendation is not to run as root. How can we engage in container image creation and usage best practices?

A: You are right! The Sysdig 2021 Container Security and Usage Report says 58% of containers run a root! Consider adding a container security scanner such as **Clair** and so you can start attacking these best practices for forming good habits early. Also, Sysdig also offers **Falco**, “the cloud-native runtime security project, is the de facto Kubernetes threat detection engine.”

Running Kubernetes Locally

Q: How is a one-node cluster working, is the master and worker node sharing one and the same node?

A: Kubernetes is really flexible and its control plane components can be manipulated for a variety of targets that are optimized for rapid experimentation or highly available and resilient production. As a developer the experience on these different targets is consistently and mostly the same. For small Kubernetes like on Katacoda, Minikube, K3s, KIND, Microk8s, and all, then etcd runs as a container, and the master and worker nodes are all just set up to be on a single node.

Q: Is it possible to install and run Minikube on Ubuntu VM running on Oracle VirtualBox running on Windows 10?

A: Have not tried. It might be possible, but you may be adding untested paths that can complicate things. If you're comfortable with hypervisors like VirtualBox, KVM, Parallels, Hyper-V then just use them directly.

Q: Does Minikube need to be installed separately?

A: Yes, it's a command-line tool. See https://kubernetes.io/docs/tasks/tools/install-minikube/

Q: I have experienced Minikube can be slow and consume memory and CPU on my Mac OS. Are there alternatives you might recommend such as Kind of k3s (version 3)?

A: Yes, there are multiple options to run Kubernetes on your laptop. Kubernetes is fairly large with its entire control plane and multiple applications can be a strain on your local resources. Especially if you add a hypervisor to it or start adding applications. So first, choose a laptop with some decent CPU, memory and I/O. If you are using Mac OS or Windows 10 professional, then KinD is built into Docker for Desktop. Just use that. There are also MicroK8s, K3s and Minishift. I would try as many as you have time for. For quick experimentation there are fast sandboxes that will not slow down your machine, such as the Kubernetes sandbox.

Q: There is now MicroK8s, it's now available on Mac and Windows in addition to Linux.

A: It uses the standard hypervisors just as Minikube does, so it should work. Frontpage says it covers all three top laptop flavors: https://microk8s.io/. I recently tried the new version this week and experienced a problem.

Q: Does Minikube have a dashboard?

A: It does, it uses the standard Kubernetes dashboard that's produced by the Kubernetes project. To see it type "minikube dashboard".

Q: Have you played with KIND?

A: KIND stands for Kubernetes in Docker. It's really easy to access if you already have Docker for Desktop already installed. I have not used it too much as it requires Windows Pro and my current laptop uses Windows Home. However, K3s, Microk8s, and Minikube are other options. (Do not confuse "KIND" with the "Kind: Pod" syntax found in Kubernetes YAML manifests.)

Q: Can you use Minikube with VMware workstation

A: Never tried, but should be just another hypervisor option. https://medium.com/@airwavetechio/running-minikube-on-windows-using-vmware-workstation-1e592872a488

Q: I have VirtualBox already. Can you list the steps used to install Minikube on Windows 10 home editor?

A: This is what I run on my laptop. Along with Git Bash in VSCode and Kubernetes plugin for VSCode. Ensure Hyper-V is turned off as it conflicts with VirtualBox. Install miniube.exe. Run "minikube start" The startup will discover VirtualBox and use it. There is nothing you need to do with VirtualBox, except for installing it. You can also just use Hyper-V on Windows Pro (not home edition). Also be sure your BIOS virtualization switch is enabled (some laptops issued by IT departments disable it, https://bce.berkeley.edu/enabling-virtualization-in-your-pc-bios.html).

Q: Why do you use Parallels and not VirtualBox on macOS? I have heard there are stability issues running Minikube under VirtualBox directly on macOS, is that why?

A: For a long time on my Windows machine I was running Minikube with VirtualBox. I like VirtualBox as a hypervisor for laptops as it's a common denominator across all three operating systems (Windows, macOS, and Linux) found on our laptops. I have nothing against VirtualBox and it has been just fine for me for years. I have seen others bash it, but have not explained why. When I was using a Mac, I wanted to try something different, just for the experience and was intrigued by the ease of use for Parallels. So I shelled out a little cash. I was pleased to start up a multi-node Kubernetes cluster on my macOS with parallels that were solid. As engineers, it's important we take the time to experiment and deviate from our happy paths when you can afford it as there is always something new to learn.

Scenarios

Q: In the scenarios, can I type them or should I click on the black text in the instructions?

A: One of the best things about Katacoda is it's hands-on with a live terminal. It's a place where you can try things and make mistakes and have successes. By reading and using your fingers to type your are reinforcing your learning process and developing muscle memory. Typing commands is a strong way for us to learn. Yes it's a little slower, and yes you will enter typoes, but your brain will be happier. However, clicking on the black command is a quicker way to get through the points. I use it when demonstrating, since none wants to watch me type, watch water boil, or watch grass grow. I will click on text when I want to skip over the boring parts I already know, then will revert back to typing where I'm learning something new or have put my locus on a specific concept. I have personally seen that concepts do not translate between my short term memory and long term memory unless I take the time to type in the commands. So, type to learn, click to move faster.

Q: In some scenarios when I attempt to access a service at a port it sometimes presents a page saying "Connecting to Port [N]. We're currently trying to connect to a HTTP service running on [N]. Services can sometimes take a few moments to start, even upto five minutes." What do I do?

A: When you start applications on Kubernetes the service address at a URL will typically be ready very quickly. In scenarios these are often NodePorts as a specific port. For instance the Kubernetes dashboard is often found at the lower NodePort number, 30000. When you first start your cluster the URL is available, but the container that services the dashboard is still starting up. If the application is still starting, then Katacoda will display this page. This page will also be shown if a container is restarting because of an error or probe failure. Before trying the URL and the NodePort, check the Pod status is the application you are trying to access, such as "kubectl get pods [-n a-namespace]". The status column should say Running for the application you are trying to access. Also, check the service to ensure the Pod has a service that is exposing it as a NodePort with "kubectl get services [-n a-namespace]". If you still cannot access the service, go back and make sure you followed all the steps in the scenario carefully, sometimes missing an instruction will cause the problem. Sometimes, albeit rarely, the Katacoda instance did not start correctly, so refresh your browser tab to invoke a new instance of the scenario to try again. Lastly, it's also possible that since the scenario was last tested, something has changed related to the dependencies the scenarios relies upon. In this case alert the author (e.g. me, jonathan.johnson@dijure.com) of the scenario or Katacoda at support@katacoda.com.

Q: How long are the Katacoda scenarios available?

A: All the scenarios at https://learning.oreilly.com/scenarios/ are available to you all the time as a subscriber.

Q: Are the scenarios from our last session gone? Seems like they are no longer here on your kata coda page.

A: They should be still there, and I will make sure they are this week. O'Reilly will soon be publishing them to the standard place for the other O'Reilly scenarios. In the cheat sheet is a list of Katacodas and the locations where they can be found. Some asked about the scenario "Decomposing Container Images", that location is also in the sheet.

Q: How did you create Katacoda scenarios for Kubernetes?

A: It's fun, easy, and free. I hope to present an O'Reilly learning session on it. You can get started here.

Q: If we do the scenarios after class will we see all your courses or only some of them?

A. You will see all of them. There are just a few that have not been published and they are still here: https://www.katacoda.com/orm-jonathan-johnson or here https://katacoda.com/javajon. For Part 2 of this series, some more scenarios will be published. I will also be converting my scenarios from version 1.14 to Kubernetes 1.18.

Q: What do I do if my Katacoda scenario is stuck, is slow, shows a strange error, or does not allow me to move forward?

A: Sometimes software does not work the way it's supposed to. These scenarios sometimes have problems. If you are experiencing a problem, jump off the lame horse and pick another one. Just refresh the browser tab and Katacoda will issue you a new instance. The downside is you will have to return to the start of the lesson, however restarting a lesson is sometimes the best way to get you out of the jam. If the problem continues, you may want to contact support at support@katacoda.com. You can also contact me a jonathan.johnson@dijure.com.

Q: There was a defect in a scenario I ran during the course. When will it be fixed?

A: I will fix it or respond to you as quickly as I can. Usually within 24 hours. However, all I ask is that you tell me what is broken or needs adjusting. Best way is just drop me a quick note at jonathan.johnson@dijure.com.

Q: For scenarios not working today, will you email us when they are working again?

A: I do not have access to your emails, but for week 3 they will all be retested and listed in the cheat sheet.

Q: I am wondering how you integrate ccat in your scenario. It is not part of the Ubuntu packages, as far as I can see.

A: In these scenarios ccat is a bash function I added. Instead of just cat, this adds syntax coloring to anything you view. I'm glad you noticed and asked, as it's a nice hack that teaches techniques to use containers on the command line. I put this in the bash startup script for the scenarios:

```
# Allow pygmentize for source highlighting of source files
(YAML, Dockerfile, Java, etc)
# Preload docker image to make ccat hot
docker run whalebrew/pygmentize:2.6.1 &
echo 'function ccat() { docker run -it -v "$(pwd)":/workdir -w
/workdir whalebrew/pygmentize:2.6.1 $@; }' >> ~/.bashrc
source ~/.bashrc
```

Q: Do we normally issue kubectl commands on the controlplane or node01?"

A: Kubectl and Helm are command-line tools that communicate to the Kubernetes Client REST API server. The tools just make REST calls to the API port, typically port 6443, you can verify the access with `kubectl cluster-info`. This means that you can run kubectl, helm, or any other application that communicates to the REST API from any machine that has network access to the cluster administration port. In the Katacoda scenarios, your bash terminal lands on the same machine as the *controlplane* node. As a developer or admin, you would be running these tools from your local machine and connecting to the cluster on your network.

Q: I would like to try these scenarios with Minikube locally, is that possible?

A: Yes, and that is another great way to learn. The installation instructions are usually fairly easy for developers. Here is how to get started: Minikube start.

Q: These scenarios are super cool. I find myself struggling to understand a few steps while doing those. Any tips? Maybe more practice?

A: Definitely these things take practice and sometimes you have to read things a few times before they make sense. The documentation a kubernetes.io is also well written and they sometimes explain topics in a better light. I always want to make sure these scenarios make sense and are clear. If they are not, or there are defects, please just send me a note or have a chat with me. There are a variety of ways to contact me here.

Learning More

Q. I have a question about something and just need a quick conversation. How can I reach you, Jonathan?

A: Just find a time on my calendar (calendly.com/javajon) for a quick chat.

Q: What are the channels you follow to keep up with the latest development in Kubernetes?

A: Kubernetes Podcast, Kubernetes slack channel, https://kubernetes.io/community/, https://kubeweekly.io/, https://kubernetes.io/blog/, and a network of friends. Most of all, start curious.

Q: Can you recommend some resources for learning more about distributed systems?

A: Distributed, parallel, grid, cloud? It's a large world of topics and I agree it may be hard to find the best resource. As an O'Reilly learning subscriber you of course have a large library at hand. Searching for "distributed, cloud, parallel, grid" evokes a nice list of books. In particular, Sukumar Ghosh's book Distributed Systems, 2nd Edition covers the breadth of topics to keep us humble. One of the founders of Kuberentes, Brendan Burns, offers a read more pointed at the Kubernetes problem domain in Designing Distributed Systems.

Q: Could you talk about Kubernetes Certification in the next session?

A: There are two types of certifications for Kubernetes Certified Kubernetes Application Developer (CKAD) and Certified Kubernetes Administrator (CKA). For CKAD I would recommend Learning Path: Certified Kubernetes Application Developer (**CKAD**) Prep Course by Benjamin Muschko. The topics in this Kubernetes in Three Week are the first steps toward preparing you to pass these certifications.

Q: Any recommendations on courses to take (including this one!) to prepare for the CKAD?

A: To train for the CKAD (application developer) certification exam I would recommend the O'Reilly training hosted by Benjamin Muschko called the Certified Kubernetes Application

Developer Crash Course (CKAD). He teaches specifically the CKAD exam and offers Katacoda scenarios to tune your skills for passing the certification.

To learn about containers, seek the teachings from Raju Ghandi.

Read the book “Kubernetes Up and Running”.

See out other thought leaders like Kelsy Hightower, Sam Newman, Joe Beda, just to name a few. O’Reilly has many books and live content on Kubernetes. Fortunately it’s a hot topic, so there is a wealth of material to learn. Most importantly install something like Minikube and challenge yourself by adding some applications you write yourself. I’m glad you chose me as part of your learning journey!

Q: Kelsey Hightower, 12 fractured apps?

A: Good reference, thank you. I mentioned that you should know 12 Factor Apps. Once you understand these factors, take a read through Kelsey Hightower's article too: https://medium.com/@kelseyhightower/12-fractured-apps-1080c73d481c. It's important to know the best practices, but sometimes it’s hard to consistently follow them.

Q: Martin Fowler was the first one, I think, to use the "you must be this tall" prerequisite comparison. https://www.martinfowler.com/bliki/MicroservicePrerequisites.html

A: During the first week I said Neal Ford has a slide about the roller coaster height. Maybe he got it from Martin, or visa versa or they are messengers of the idea from someone else. I'll ask Neal and see if we can get to the source. However, notice it's talking about being tall enough for microservice but I used it for being tall enough for DISTRIBUTED microservices on a cluster. This is also just a reference to the idea of knowing your team's maturity model. Not being tall enough will compromise your Kubernetes success.

Q: Can you change the format you save your slides in so we can click on links in it or search the slides? Right now you can't do either.

A: I agree the slides are missing this ability to click on links. With the live Prezis we can click on the links, but not in PDF form. I will provide a list of referenced URLs in the cheat sheet. I use a presentation software called Prezi and I have asked them for this feature. So far they have not delivered. I will see if I can provide URLs to the live Prezis for you all.

Q: Some pages in the PDF for the slides appear to be clipping part of the slide on the sides. Where can I see the rest of the slide?

A: These slides are exported from the presentation program called Prezi. The views you are seeing of each PDF page are landing places when I am walking you through various scenes in the presentation. When images on a page appear clipped that means the view is zoomed or panned into an area of interest. Just ignore the clipped parts and focus on content in the middle of the PDF page. The full, unclipped image of the Prezi page appears in previous PDF pages.

Q: Is this session recorded? How can I see previous weeks?

A: As a subscriber of O'Reilly learning and with your registration for this Kubernetes learning series you will see all the content for the weeks you register for. The content improves over time and some early technical issues have been resolved so future sessions covering the same material will only improve. Our software evolves the same way, usually. You can register for future times of Part 1 and Part 2 to get access to that new material. The schedule for the 2020 sessions are on the cheat sheet. Just return to the URL when you attended the course and all the material will be there, including this Q&A. The session recordings remain available for up to a year.

Q. Where is the survey for the course?

A The survey is accessible through the widget tray towards the bottom of your screen.

Q: I see there are three parts for Kubernetes in 3 weeks. I'm only registered for one part. Where do I register for the other parts?

A: Each part is 3 weeks and there are three parts for a whopping total of 9 weeks of information. That's a total of about 30hours of learning Kubernetes. I would encourage you to sign up for all 3 parts. Go here (https://learning.oreilly.com/attend/) and search for "Kubernetes in 3 Weeks" to see what parts are currently scheduled.

Q: What topics do you cover in the other O'Reilly trainings?

A: A bunch....

Part 1 · Fundamentals

Week 1: Getting Started with Kubernetes	Intro, terms, architecture, commands, 1st apps
Week 2: Containers and Microservices	Packaging, best practices and running apps
Week 3: Pods and Common Objects	Best practices forming Pods and other key objects

Part 2 · Intermediate

Week 1: Extensibility and Operators	Extensibility and architecting operators
Week 2: Service Meshing	Network traffic control, security, observability
Week 3: CI/CD Pipelines	Clusters for building, testing and delivery models

I have other upcoming online training on these topics as well:

Microservices testing	Consumer-drive contracts and Chaos testing
Observability	Logging, tracing, metrics, events, and dashboards
Serverless	Rapid delivery, functions, knative architectures

Q: Would it be possible to extend next week's session from 3 hours to 4?

A: Each week covers a central theme. The Kubernetes Fundamentals, (week 1, part 1) and CI/CD pipeline with Kubernetes (week 3 of part 3) are 4 hours, the rest are 3 hours. I understand there is much to cover and going too fast can be frustrating. I opt to go faster since you can always review the recorded session and re-try the scenarios at any time.

Q: Will there be an Admin course?

A: There are many more topics I want to cover in this O'Reilly learning series. There are lots of interesting topics about operations of a cluster, especially on "day-2+". I'm proposing a few more topics and some optics cover administration. Keep in mind sometimes Administration get into specific cloud implementations, and I like to keep my material generic to Kubernetes itself. If you have more topics you want me to cover, just drop me a quick note.

Q: A few of you asked about that Fondue recipe found in the scenario Rust to Kubernetes scenario. Here it is. It's a great social meal to gather for with other vaccinated friends. Let me know how it turns out!

A: Fondue recipe

You'll need a pot, something you can mount over a small flame like a few tea candles, a sterno can, campfire, etc. Various garage/tag sales or flea markets invariably have "fondue" pot sets from the 70's. You can also pick up a new fondue pot as it's a nice investment for social gatherings. Make sure it comes with a heat source and those long forks.

- Heat pot on stove - between medium and low
- Shred 1 pound **Emmental** cheese and ½ pound **Gruyère** cheese - from a good cheese shoppe
- Add about 5 oz of dry white **wine**. Take a few ounces for yourself as well.
- Add 2 to 4 crushed or chopped **garlic** cloves
- Gradually add cheese and stir

- You must stir in one direction only. If you don't, the Imps of the Swiss mountains seek the disturbance
- Stir the melt for 10 minutes
- Add mixture of 1 oz **kirsh** + 1 tbsp **cornstarch**
- Stir in and transfer pot to a heated fondue stand where **friends** are gathered.
- Serve immediately with long forks to dip: **french bread** cubes. Other tasty dipping nucleuses can include ham, carrots, braised broccoli, colliflower, apple chunks, steak tips, hot peppers, anything else you can dream up.
- If you are very social, you can double the ingredients, but prepare as two batches.
- Next day, but before the visit to the cardiologist, the leftover cheese makes an amazing cheese type casserole such as with veggies, pasta, and left over dipping ingredients. Mac and cheese. Cheese omelette.

Q: What tools do you use for presenting?

A: Prezi for slides. Open Broadcasting Software (OBS) for streaming. Powerpoint Isometric transforms for 3d creating diagrams. I use Hugo and GitHub pages for my www.dijure.com.

Q: Do you offer other training?

A: Yes, I travel and present online for No Fluff Just Stuff, a Software Symposium Series. I travel with my fellow speakers throughout the U.S. and now with the COVID-19, there are lots of great presentations. I also contribute to a few other conferences: Global Big Data, Software Design, and Development in London, Great International Developer Summit in India and Australia, and a few more.

I'm available for private training and consultation as your team dives into the challenges of cloud native, container, microservices, and of course Kubernetes. Sometimes I team up with Mark Richards and other experts with courses on Microservices architecture, containers, and other interests your team is finding challenging. Here is my contact information at https://www.dijure.com/contact/