{mthree}

Kubernetes

Site Reliability Engineering

Overview

Learning Objectives

In this module, you will be introduced to the deployment system

By the end of this module, you will be able to:

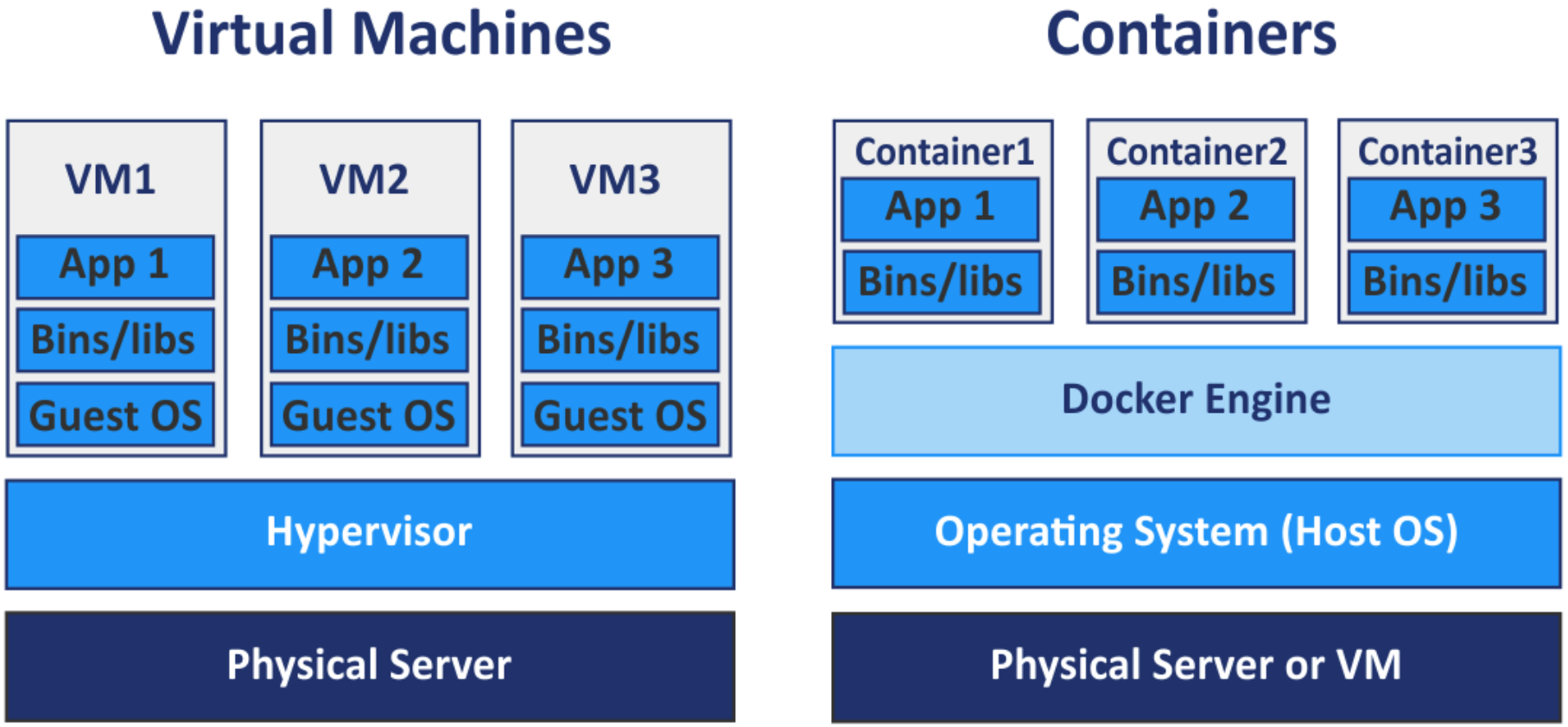
- Explain what a container service is
- Review and modify infracode
- Debug deployment issues
- Explain the difference of create vs. update

Kubernetes

- A container orchestration tool to manage enterprise scale microservices
- Manages
  - >>> Container infrastructure (networking, load balancing)
  - >>> Deployments
  - >>> Persistent Storage
  - >>> Secrets
  - >>> Variables

{mthree}

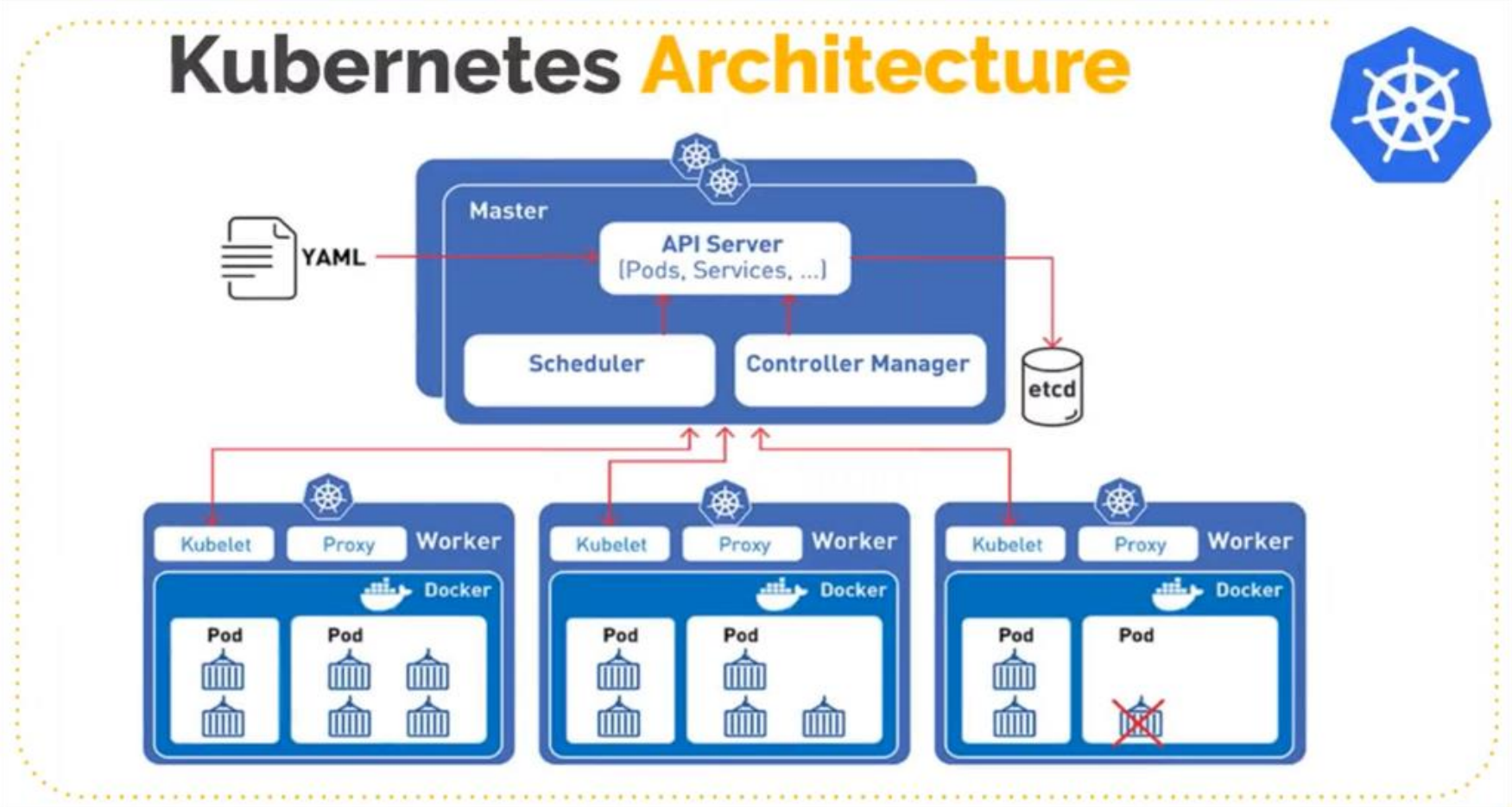
VMs vs Container Deployments



Source: Docker containers are not lightweight virtual machines

{mthree}

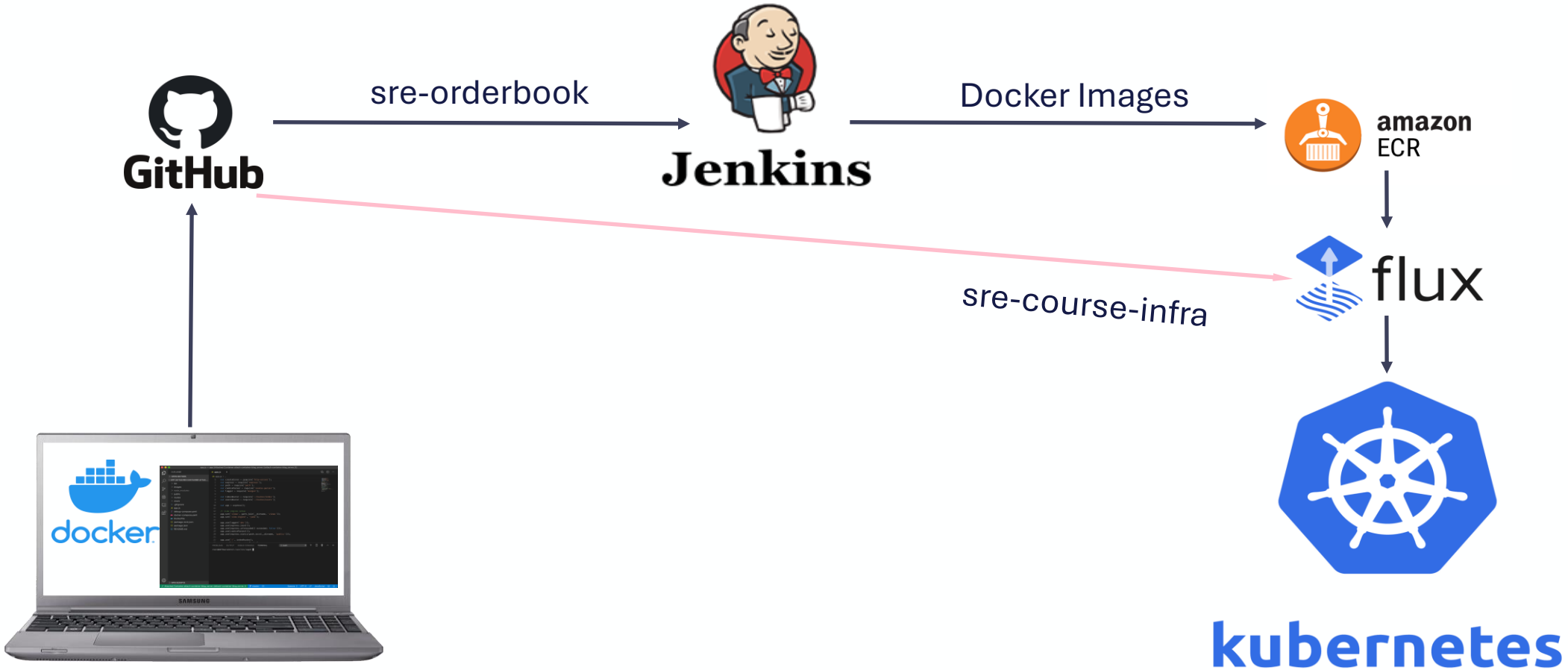
Containers in Kubernetes



Source: Getting Started With Kubernetes - Part Two

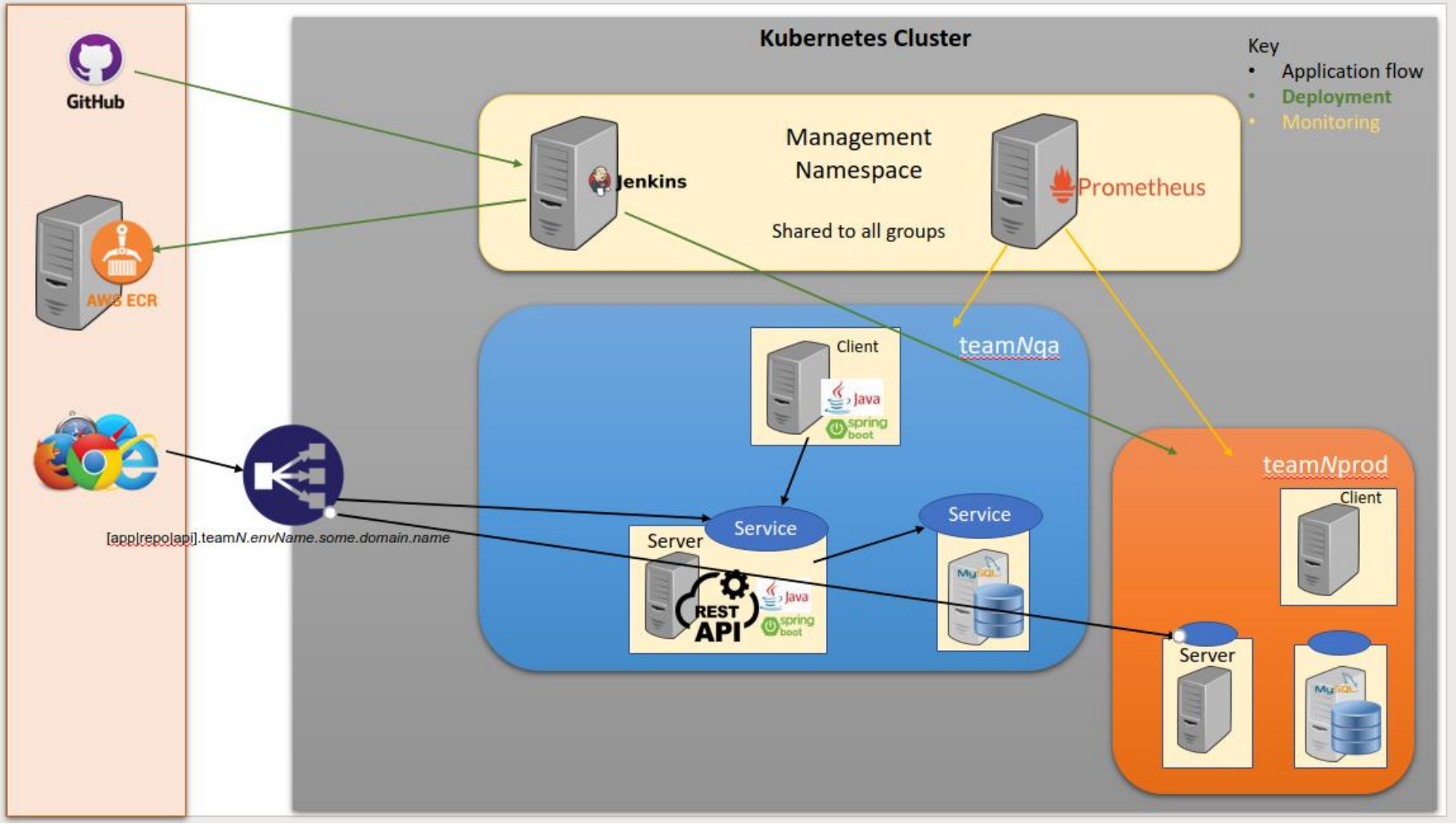
{mthree}

Containers to Kubernetes



{mthree}

Our Lab Environment



{mthree}

Services of Help

- Jenkins
  >>> https://jenkins.computerlab.online
- Grafana Monitoring Dashboard
  >>> https://grafana.computerlab.online
- Prometheus metric gathering
  >>> https://prometheus.computerlab.online
- Alertmanager
  >>> https://alertmanager.computerlab.online
- Docker registry repository
  >>> http://ecrlist.computerlab.online/index.php
- Kubernetes
  >>> https://k8sdashboard.computerlab.online

{mthree}

The Lab Repository

- Kubernetes is managed through
  - >>> GitHub.com/The-Software-Guild/sre-course-infra
- DevOps practices must be used with this repository
  - >>> Coding on feature branches
    - ~ Never code on main
  - >>> Pull requests required to merge your code to main branch

Activity: Kubernetes – Step-by-Step

IMPORTANT: Naming convention!!!!
Use lowercase always!!!!

- Create Dev and Production environments
  - >>> Using the sre-course-infra Git repository and the make-sre-env script
    - ~ This is a one-off task for each environment
  - >>> Make sure you use a branch to do your updates
    - ~ Branch name to follow the format of ?XXXteamNN
      (? = r – reskill c – cohort)
- Deploy Dev
  - >>> Use PR in Github web page to merge your changes to main
  - >>> FluxCD will detect your changes to the sre-course-infra and create the environment, or will identify new Docker images in the ECR and deploy
- Promote to Production
  - >>> Make a copy of the Hardcoded Promote Jenkins job
    - ~ Change orderbook in the environment section to your ?XXXteamNN Jenkins job name
      (? = r – reskill c – cohort)
  - >>> Modify to create your own Production image
  - >>> Or view https://github.com/The-Software-Guild/sre-course-examples
    - ~ Branch d1m7t1
    - ~ Directory Day1/Module7/Task1/instructor-promote-prod
    - ~ Change instructor to your ?XXXteamNN Jenkins job name
      (? = r – reskill c – cohort)
- Deploy to Production
- View the logs in Grafana to detect any errors
  - >>> Use {job="flux-system/kustomize-controller"} using Loki to determine if there are any errors in your sre-course-infra code

{mthree}

Activity: Creating Environments – Step-by-Step

This is a one-off task!

- Create Dev
  - >>> Clone the following repository
    - ~ https://github.com/The-Software-Guild/sre-course-infra
  - >>> Create a new branch called ?XXX-teamNN (? = r – reskill c – cohort)
    - ~ Where XXX is your course code (instructor can help if you do not know)
    - ~ NN is the team number assigned by your instructor
  - >>> Run the script called make-sre-env
    - ~ You will need to supply the following as command line arguments
      - Course in the format of ?XXX (? = r – reskill c – cohort)
      - Team in the format of teamNN
      - Environment to create dev
      - Example - ./make-sre-env c149 team01 dev
  - >>> Add, commit and push your branch
  - >>> Create a Pull Request to merge to the main branch
  - >>> Merge if PR allowed, or fix conflicts then come back to merge
- Create Prod
  - >>> In the same git repository run make-sre-env again, but change dev to prod
  - >>> Perform the same actions as above for getting your deployment actioned

IMPORTANT: Naming convention!!!!
Use lowercase always!!!!

{mthree}

Activity: Changes to Environment – Step-by-Step

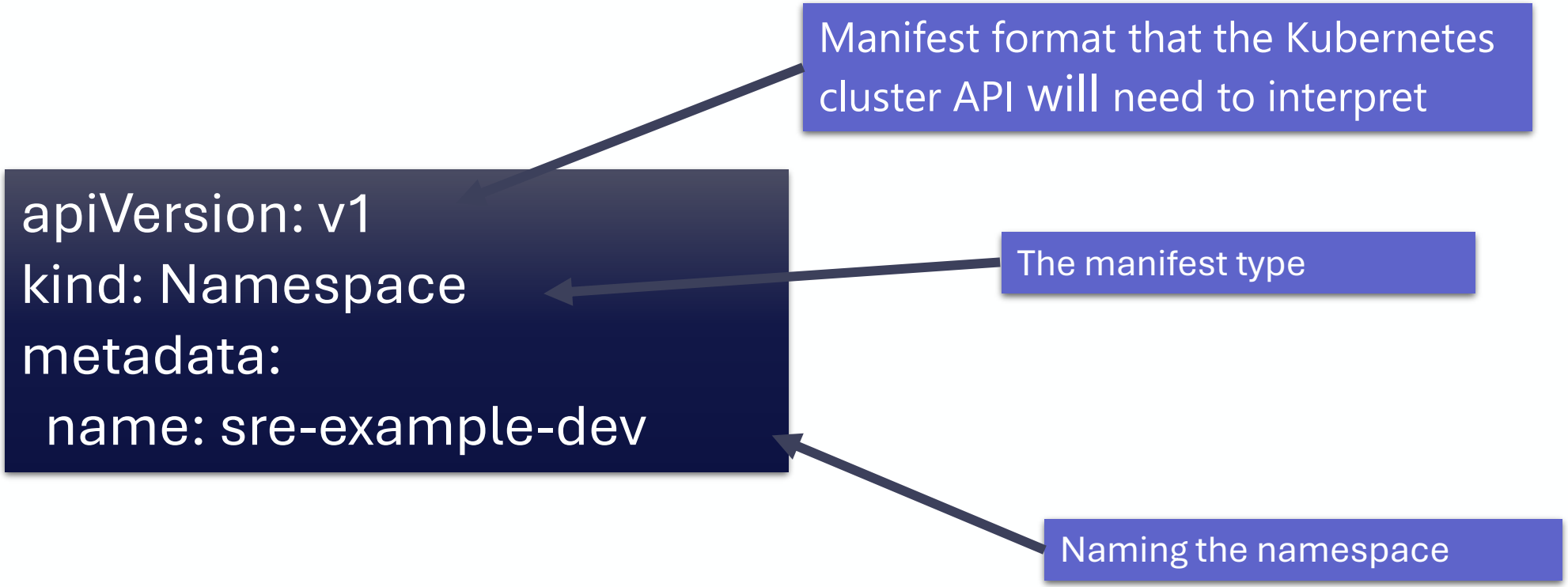
- Any changes should be done on your branch
- Follow the add, commit, push, and pull request method to release
- Always check that there are no deployment errors in Grafana/Loki
- Where did make-sre-env put my manifests?
  - >>> **flux/apps/eks-sre-course/?XXX-teamNN-[dev|prod]**
    - ~ Use this location for any changes you wish to make to your application infrastructure
    - ~ All actions for Kubernetes to perform
      - e.g., deployment modification, DNS name for ingress, service names, etc

{mthree}

namespace.yaml

Created by the make-sre-env script

- This defines the unique place for all your application components in isolation



Access from the WWW – ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: orderbook
  namespace: sre-example-dev
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  rules:
    - host: sreexampledev.computerlab.online
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: orderbookapi
              port:
                number: 8080
```

Namespace to be applied to

External service that knows us; Required for HTTPS connections

Domain name to access this service

URL to get to the service

Service name and port in Kubernetes

{mthree}

Manifests

- Define resources in Kubernetes
- Kinds of manifest
  - >>> Namespace
  - >>> Deployment
  - >>> Service
  - >>> Ingress

{mthree}

Activity: Checking the App

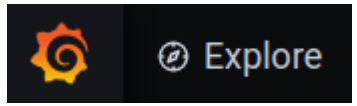
- The application:
  http://?XXXteamNNdev.computerlab.online
- We can also check the status
  - >>> By clicking the **View App Status**
  - >>> By going to http://?XXXteamNNdev.computerlab.online/status
- Click **View Order History** link to view some preloaded transactions.

NOTE: Your URL can be found in the **ingress.yaml** file value of the **host** attribute

{mthree}

Activity: Checking the Logs

- Check that Prometheus and Grafana have picked up our environment
  - >>> Navigate to https://grafana.computerlab.online



- Here we want to Explore
  - >>> Click the pull-down to the right of the Explore button and select Loki if it is not already selected.
  - >>> Below this there are some query options
  - >>> Click the pull down called Log labels and select namespace
  - >>> Scroll down to find your namespace which should be named team??dev
- Click the Run Query button, top left
- You should see the logs relating to your namespace for the containers

{mthree}

Check for Deployment Errors

- Errors can break the build for everyone
  - >>> You need to check the logs
  - >>> Loki query `{job="flux-system/kustomize-controller"} |= "error"`
    - ~ This can be used to determine if there are errors in Flux deployments
- LogQL – used by Loki in Grafana
  - >>> LogQL in Grafana Loki

{mthree}

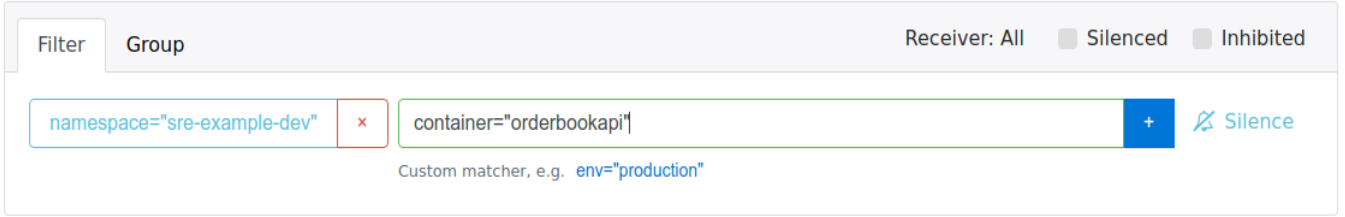
Activity: Kubernetes Failing Application

- Make a deployment fail if it hasn't
  - >>> We can do this by making a typo in the image name
    - ~ Edit the deployment-api.yaml file
    - ~ Locate image line
    - ~ Change *orderbookapi-dev* to *notebookweb-dev*
    - ~ Remove the entire comment following the image value
    - ~ Add, Commit, Push, PR merge
    - ~ Check k8sdashboard and grafana for errors, and alertmanager
- Review the failure in Grafana and Prometheus
- Return production to normal
  - >>> Remove the typo: Recreate from the api.yaml file
- Prevent failure and reliable deployments
  - >>> Kubernetes deployment strategies

{mthree}

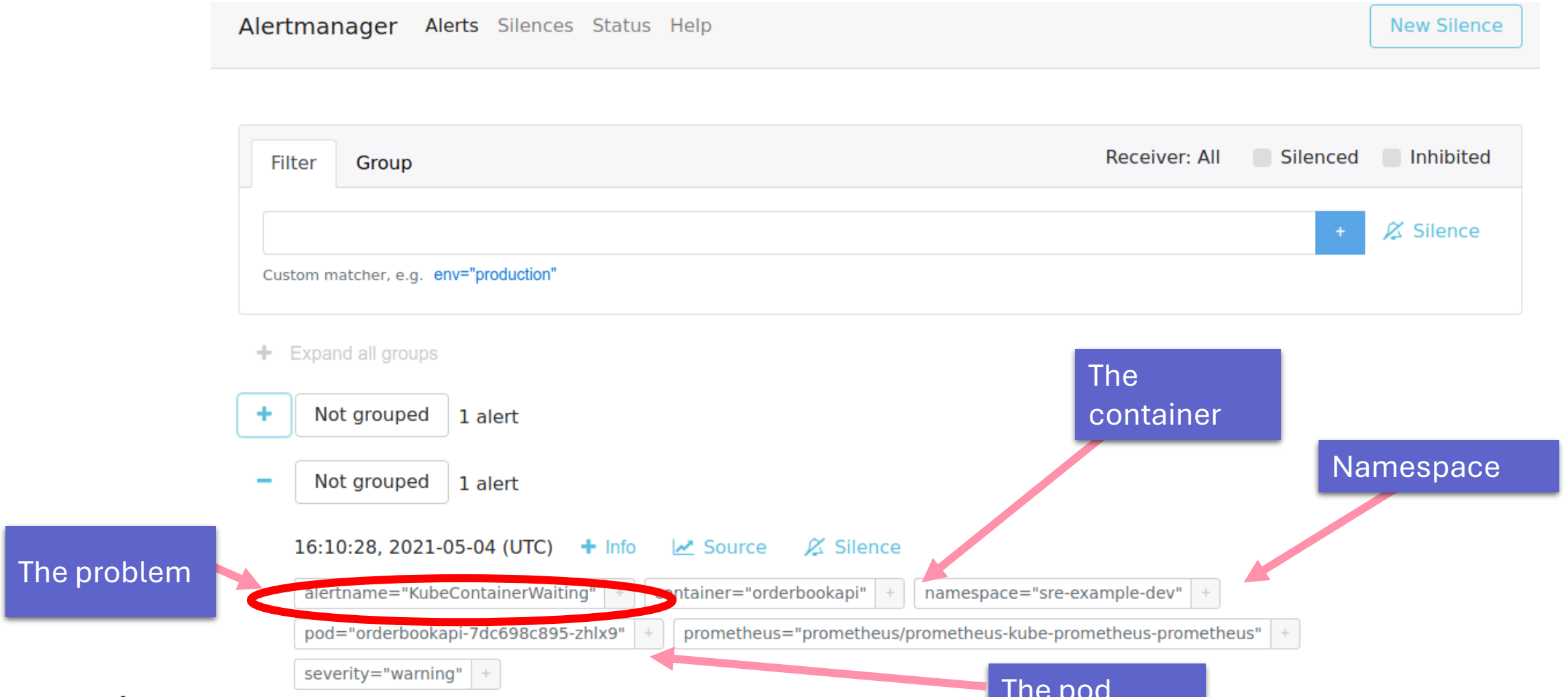
Activity: View the Error

- Open https://alertmanager.computerlab.online
  - >>> This is our alerting system when problems occur
  - >>> If there is an issue with your pod launching, you will find it here
- In the filter, you can type things such as
  - >>> namespace="yourNamespace"
  - >>> container="yourContainername"



{mthree}

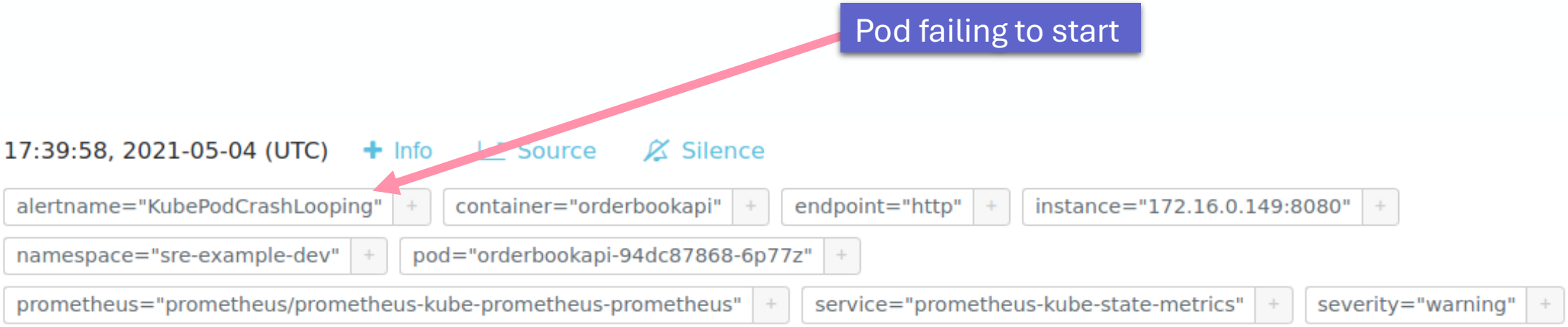
Identify the Error



{mthree}

Alert: CrashLoop

"Application fails to run" generally causes this issue



{mthree}

Deployment Manifests

Deployment kinds

- Define how your containers are to be deployed into the environment
- Name of the container/deployment
- Variables to be passed
- Ports to expose for the service to be used
- The image to be deployed
- How many to launch

References

>>> Understanding Kubernetes Objects
>>> Kubernetes: Deployments
>>> Kubernetes: Managing Resources
>>> Kubernetes API
>>> Kubernetes: Deployment

{mthree}

Preventing Failure

- RollingUpdate
  - >>> Create a ReplicaSet for the deployment
  - >>> The ReplicaSet is tested (if applied) to ensure it works
    - ~ If the pod dies, the replicaset is destroyed
  - >>> The original ReplicaSet remains running until the new one works
    - ~ Allows existing connections to be completed (pool draining)
- Create
  - >>> Destroys the current ReplicaSet and then launches the new one
    - ~ All existing pods/containers are destroyed immediately
- Useful reading
  - >>> Kubernetes deployment strategies

Pros/Cons

Create

- Pros
  - >>> Application state entirely renewed
- Cons
  - >>> Downtime that depends on both shutdown and boot duration of the application

RollingUpdate

- Pros
  - >>> Version is slowly released across instances
  - >>> Convenient for stateful applications that can handle rebalancing of the data
- Cons
  - >>> Rollout/rollback can take time
  - >>> Supporting multiple APIs is hard
  - >>> No control over traffic

Blue/Green Deployment

- Create and RollingUpdate are pod-based
  - >>> Update allows for a slow roll out and protects current connections
- What if our old API will cause issues with the new roll out?
  - >>> For example, a missing field in the API when updating the database
- Blue/Green deployment
  - >>> Ensure the load balancer sends traffic only to the new version
  - >>> The service is updated and uses a version label in the spec.selector section
  - >>> The deployment will also need to have a version label for matching

Blue/Green Example

```
apiVersion: v1
kind: Service
metadata:
 name: my-app
 labels:
  app: my-app
spec:
 type: NodePort
 ports:
 - name: http
  port: 8080
  targetPort: 8080

 # Note here that we match both the app and the version.
 # When switching traffic, we update the label “version” with the appropriate value, ie: v2.0.0
 selector:
  app: my-app
  version: v1.0.0
```

{mthree}

Blue/Green Pros/Cons

Pros

- Instant rollout/rollback
- Avoid versioning issue, change the entire cluster state in one go

Cons

- Requires double the resources
- Proper test of entire platform should be done before releasing to production
- Handling stateful applications can be hard

Canary Deployment

- Running multiple versions at the same time
- Good for user testing
- Requires two deployments and more management
  - >>> Monitoring of traffic and pods required
  - >>> Need to know which users are working with which version
- Defined using a percentage split (e.g., 75% old – 25% new)
- Service uses **app** label selector, no version
- Deployments use same label for app
  - >>> Different version label
  - >>> Different number for **replicas**, e.g., 3 and 1

{mthree}

Canary Pros/Cons

Pros

- Version released for a subset of users
- Convenient for error rate and performance monitoring
- Fast rollback

Cons

- Slow rollout
- Fine tuned traffic distribution can be expensive
- More detail in monitoring
- Complex support
  - >>> Need to know which version users are using

{mthree}

Other Features – 1

The deployment is the key element

- Look at the features that the deployment offers
  - >>> Strategy
    - ~ Allows defining of the type of deployment, rolling, recreate
  - >>> Replicas
    - ~ The number of containers to start
  - >>> MinReadySeconds
    - ~ How long to wait to ensure container isn't crashing
- Pod template spec parameters
  - >>> Volumes
    - ~ Enable persistent storage
    - ~ Configure a Pod to Use a Volume for Storage
  - >>> RestartPolicy
    - ~ Whether to restart the container if it fails
  - >>> Liveness and Readiness probes
    - ~ Determine if the container is still running and ready to run using a status check
    - ~ Configure Liveness, Readiness and Startup Probes

{mthree}

Other Features – 2

- Secrets
  - >>> Passwords
  - >>> SSH Keys
  - >>> SSL Certificates
- Add through
  - >>> secretGenerator yaml files
    - ~ Managing Secrets using Kustomize
- A secret manifest
  - >>> Secrets

```
secretGenerator:
- name: db-user-pass
  literals:
  - username=admin
  - password=1f2d1e2e67df
```

```
apiVersion: v1
kind: Secret
metadata:
  name: db-user-pass
type: kubernetes.io/basic-auth
stringData:
  username: admin
  password: 1f2d1e2e67df
```

{mthree}

Other Features – 3

- ConfigMaps
  - >>> Allow storing of variables and values for use in deployments
  - >>> Pods/Containers may require variables to be set
  - >>> ConfigMaps
- ServiceAccounts
  - >>> Used to allow pods to interact with Kubernetes services
  - >>> Configure Service Accounts for Pods

{mthree}

Summary
Q & A

{mthree}