Part 1

Write a brief go live plan – no need to be extensive, details will be discussed at the final interview. This should include:

- **How the code build would work**

1. Get the app code from developers checked/initiate into a SCM repo such as Github for e.g: by starting off with a master branch initially and then do git checkout as a feature branch for any regular modifications / future release changes.
2. Once all good with feature branch, get the code merged into a master branch by going through a Pull Request process and get it approved by other colleagues with in a team as a 2nd pair of eyes to ensure code quality is good from code perspective.
3. As part of the above process, docker image will be built using dockerfile which will make use of application binaries and copying it to the application as a docker image. as a security best practice run the Dockerfile with non-root user.
4. It will then be pushed to docker registries such as gcr (private registries preferred)/ docker hub with proper version tagging as a best practice rather than just mentioning as a latest tag.
5. Later on, the specific tagged docker image will be used in kubernetes manifests as a deployment file. Here its good practice to split services as individual files for each deployment, service, PVC and other related files which are essential as part of application deployment.

- **How the deployment would work**

1. Create individual resource definition for objects such as deployments, services, PVC, configMap, secrets, namespace etc.
2. once you got all the related manifest files created and ready for deployment.
3. Create a dedicated namespace for each projects to have a logical separation between different apps
4. It can be run using **kubectl apply -f .** by specifying the directory path where there manifests files are located.
5. It can be made easy further with other tools such as Kustomization by specifying all deploy files in one main kustomization.yaml

secretGenerator:

- name: mysql-pass

literals:

- password=random123

resources:

- mysql-deployment.yaml

- wordpress-deployment.yaml

- redis-deployment.yaml

1. As a final standard and extensive deployment procedures, Helm can be used to do above deployments. [Helm is a package manager, release manager and a templating engine.](https://learnk8s.io/helm-templating-kubernetes-yaml) Helm will have file structure similar to below where The values.yaml file contains all the customisable fields

tree

.

├── Chart.yaml

├── templates

│ └── pod-template.yaml

└── values.yaml

Helm is usually a popular choice because you can share and discover [charts, its sa collection of Kubernetes resources](https://helm.sh/docs/topics/charts/).

- **How configuration and credentials would be passed to the applicatio**n

1. Its always recommended as a best security practice to include securitycontext and pod security policies as part of creating above resources definition.
2. Especially app configuration needs to be created as configMap and senstivie information such as credentials would need to supplied in pods/ deployment manifests files as Secrets.
3. It can also be integrated with 3rd party tools such as vault etc to store this credentials, certificate info and make use of these vault ids in resource definition files.

- What monitoring you would put in place.

[Prometheus](https://prometheus.io/) and [Grafana](https://grafana.com/) are some of the most popular monitoring solutions for Kubernetes, and they now come as default in most managed clusters such as GKE. Both of these solutions can be installed using Helm charts which are public kubernetes charts and needs adding to local helm repos and customised accordingly to our environments.

As a best practice, it needs installing in a separate namespace such as monitoring. Ensuring both prometheus and grafana pods were successfully running and installed. As part of this monitoring solution installation with Helm, it will also deploy tools such as alertmanager and node-exporter which will help in monitoring working along with prometheus and grafana pods.

Kubectl get pods -n monitoring

NAME READY STATUS RESTARTS AGE

Alertmanager-prometheus-prometheus-oper-alertmanager 2/2 Running 0 16m

prometheus-grafana-5c5885d488-b9mlj 2/2 Running 0 19m

prometheus-kube-state-metrics-6967c9fd67-zsw6c 1/1 Running 0 19m

prometheus-prometheus-node-exporter-jj6wq 1/1 Running 0 19m

prometheus-prometheus-oper-operator-77cbbc55f5-6btf2 2/2 Running 0 19m

prometheus-prometheus-prometheus-oper-prometheus-0 3/3 Running 1 15m

**Prometheus** can be used for logging using prom query and **Grafana** will be used as a GUI Dashboard tool with prometheus as a datasource. With Grafana as GUI Dashboard tool, we can create multiple Dashboards with different panels for monitoring custom metrics in each Dashboard as required. All the custom monitoring can be specified with Prometheus scrape configuration as different CRD files while setting up this monitoring.