DOCTORLY TECHNICAL ASSESSMENT

Contents

1 Task #3: Monitoring Production Servers

1.1 Introduction

When faced with the task of monitoring production Ubuntu instances, three important considerations come to mind for me:

1. Choice of tools & technologies to use.
2. How to implement the chosen technologies as a running system
3. Setting up proper channels to alert the relevant administrators of any possible issues.

Below, I will briefly discuss each of these in more detail.

1.2 Monitoring Tools & Technologies

I have created and configured metrics & monitoring previously, using a *Prometheus-Grafana* stack (and in one case an additional *blackbox_exporter* component for monitoring legacy web and database servers), and this has worked extremely well.

1.3 Implementation

My implementation of choice for the monitoring stack would be

- Deploying the monitoring stack in Kubernetes.
- Using Helm charts for deploying the different components.

Alternatively, I'd been considering the creation of a Docker cluster (docker compose) for the monitoring stack if Kubernetes or equivalent was not available to me.

1.4 Alerts

The third consideration would be to set up instant alerting with high visibility.

In my experience email alerts, while useful for detail analysis & troubleshooting, lacks immediacy: People tend to ignore emails or they are simply too occupied with other work to notice an email alert in reasonable time.

I would therefore prefer to set up an instant alert channel using tools such as *MS Teams, Opsgenie, etc.* together with email alerting.

I think it is also very important that the metrics set up in Prometheus & Grafana, should be relevant, accurate, and as easy as possible for a viewer to understand; hand-in-hand with this, I would take care that alert messages are accurate and easy to understand, rather than vague and/or misleading.

2 Task #4: A Basic CI/CD Pipeline

2.1 Introduction

In my current role as DevOps Engineer, I have created and updated a *GitLab* CI/CD pipeline for deploying API's and a website, and I would follow the same approach here.

2.2 Approach & Assumptions

1. The application is an API written in *Typescript/Javascript*; I will call it **api-fdt**.
2. The *Rush* build tool is used to build, run tests, and publish the application.
3. The application is published a as Docker image that is pushed to the GitLab container registry, from which it may be pulled by a Helm deployment into Kubernetes.
4. The `package.json` file for the application contains a set of scripts to manage building, testing, publishing, and pushing the resulting Docker image to the GitLab container registry.
5. The CI/CD pipeline will have 3 stages:
   - Build
   - Test
   - Publish
6. The pipeline is configured in the *gitlab-ci* yaml file.

2.3 Reference Material

In the `task-4/` directory I have included some relevant files - extracts that still need to be expanded and tested, but could serve as scaffolding and a starting point for the pipeline implementation.

package.json: Containing scripts that could be executed by `rush` or `rushx` to produce the end-product.

dotgitlab-ci.yml: Renamed `.gitlab-ci.yml` for the sake of visibility; this is the GitLab CI/CD pipeline configuration file.

bash helper scripts: Some example helper scripts (in the `task-4/helper-scripts/` directory), that are being called by some of the `package.json` scripts.

readme.md: A basic document explaining to users how the CI/CD pipeline works.

3 Task #5: Helm Chart for a WordPress Application

3.1 Introduction

In my current role I have actually done several WordPress deployments into AWS Kubernetes.
For this task I would like to submit an example Helm chart based on one of these deployments.

3.2 Description

1. The chart is based on *WordPress packaged by Bitnami*.
2. Unfortunately, due to me running out of time in this assessment, I haven't been able to do a real deployment and test it.
3. I have, however, included my modified chart and values file in the directory `task-5/helm/`
4. I also include the output from the commands below in the directory `task-5/output/`

- Rendered templates:
  ```
  helm template -f values.yaml chart/wordpress |& tee helm-template.txt
  ```
- Install dry-run:
  ```
  helm install --dry-run -f values.yaml doctorly-demo chart/wordpress |& tee helm-install-dry-run.txt
  ```