Project:
Plan, Reduce, Repeat

Directions and Submission Template

Duong Hien Hong Thach
May 03, 2023

Overview:

You have recently joined the SRE team for an exotic plant reseller startup. They already have a small SRE team in place consisting of two other members. You are just finishing up your training period and are now ready to be on your own.

You have a busy week ahead of you as there is a release this week plus your on-call shift. Part of your release duties includes helping to maintain the as-built document by adding this new release, as well as planning for system resource changes. For your on-call shift, you have to respond to alerts as they come in and write up an on-call summary to document your shift. Finally, you'll round out your week by helping to reduce toil. You will have to identify any toil you encounter throughout the week and create a toil reduction plan. After you have a plan all ready, you will need to work on implementing that plan by writing some scripts to help automate tasks.

Scenario 1

Release Day

Release Night

Summary

Tonight is release night, and it will be your first time assisting with a release as an SRE. The process now is manual, with no real consideration for how releases may impact resource allocation. Luckily, your other team members have started implementing an as-built document. You'll have to add tonight's release to the document. The release is a pretty major release with the addition of a new feature that will bring in a large number of new clients. Looking at the results from testing, you can see that this new feature is going to add additional resource requirements as it is both more memory and, to a lesser extent, CPU intensive than before.

Current Release Features

This release will have the following changes that will need to be documented on the as-built design document. The developers have been hard at work implementing the following tickets:

- Ticket 203 added a new catalog for exotic plants. This ticket added new tables in the database to handle the additional catalogs.
- Ticket 202 rearranged the catalog menu in the UI to accommodate the additional catalog, as well as making it more user-friendly.
- Ticket 201 added an additional component to the application, an order processor. The order processor is responsible for batch processing orders on a schedule. The reasoning behind this was to decouple the UI from order processing, and since order processing can be CPU intensive, this decoupling prevents the app from performing poorly. The Design Doc 5247 goes into more detail about the design specifics.
- Ticket 205 fixed a security flaw where attackers could execute a SQL injection attack.

Release Night, cont.

Release Process

The established release process is a manual affair generally done by one of the operations team members. The OPs team generally will download the latest code, shut down the app, run the database migrations, change or add any needed configurations and then start the app back up. In the past this has caused issues as steps have been forgotten, not all the scripts were executed, the app was not restarted properly, among other issues. During the release window, the OPs engineer would also add new resources as needed. This has led to downtime in the past as the app became overloaded and could not serve requests anymore.

Release Planning

During load testing for this release, it was determined that

- Main Application
  - The new catalog feature increases RAM usages by 25% for the same number of users, while not increasing CPU significantly. Currently, the main application containers are utilizing almost 85% of the RAM allocated.
  - At the current resource allocation, each server can handle 500 concurrent users. Currently, there are 3 application containers to support about 2000 total users, with about 1300 being on at any one time. This release is expected to add about 1.5 to 2.5 times the total number of users, with a need to handle 2600 users concurrently.
- Order Processor
  - This component has a high CPU utilization with moderate RAM requirements. In testing, a full loaded queue used approximately 1 Gb of RAM.
  - The component runs with 2 concurrent processes, pulling orders out of the database and processing them for fulfillment. This component can process 4 orders at a time, with the average order taking between 10 and 15 seconds to complete depending on the size and complexity of the order as well as CPU resource allocation. QA recommends twice the CPU as the main application.
- Database
  - The database was provisioned to handle a much larger application than what the company has now and passed the load tests with flying colors.

As-Built Doc
Release 1

Stakeholders

- Developers
  - John Doe
  - ane Peters
  - Sam Ross
- Ops
  - Jay Smith
- SRE
  - John Robert

Code Changes

- Security fixes
  - Added new password requirements (Tk-100)
  - Fixed how SQL queries were handled (Tk-103)
- Feature Additions
  - Added new menu options for users (Tk-102)
  - Users can now have middle names (Tk-101)

Data and System Changes

- Data model changes
  - Added columns for middle names in user table (TK-101)
  - Added additional New Menu table (Tk-102)
  - Users table was split into 2 smaller tables (TK-101)

As-Built Doc
Release 1

Design decision highlights

Users table was split into two smaller tables to create more efficient queries and mappings. Keeping it as one big table began to cause slow queries and allowed for a larger number of users. See Design Doc 134 for further discussion.

Test Section

All test suites are passing 100%.

Deployment Notes

The database admins asked for an additional set of scripts to be run for data corrections.

Deployment File
Release 1

```
ApiVersion: apps/v1
kind: Deployment
metadata:
  name: app-deployment
  namespace: course4
  labels:
    app: mainApp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mainApp
  template:
    metadata:
      labels:
        app: mainApp
    spec:
      containers:
      - name: mainApp
        image: nginx:latest
        resources:
          requests:
            memory: 256mb
            cpu: 250m
        ports:
        - containerPort: 80
```

As-Built Doc
Release 2

Stakeholders

- Developers
    - John Doe
    - ane Peters
    - Sam Ross
- Ops
    - Jay Smith
- SRE
    - Thach Duong

Code Changes

- Security fixes
    - Fixed SQL injection attack (Tk-205)
- Feature Additions
    - Added a new catalog for exotic plants (Tk-203)
    - Rearranged the catalog menu in the UI (Tk-202)
    - Added new component order processor (Tk-201)

Data and System Changes

- Data model changes
    - Added new tables for catalog (Tk-203)

As-Built Doc
Release 2

Design decision highlights

The order processor is responsible for batch processing orders on a schedule. The reasoning behind this was to decouple the UI from order processing, and since order processing can be CPU intensive, this decoupling prevents the app from performing poorly. See Design Doc 5247 for more detail about the design specifics.

Test Section

Here are some highlights from load testing:

- Main Application
    - The new catalog feature increases RAM usages by 25% (are utilizing almost 85% of the RAM allocated).
    - Need to handle 2600 users concurrently.
- Order Processor
    - High CPU utilization with moderate RAM requirements. In testing, a full loaded queue used approximately 1 Gb of RAM.
    - This component can process 4 orders at a time, with the average order taking between 10 and 15 seconds to complete depending on the size and complexity of the order as well as CPU resource allocation.
    - QA recommends twice the CPU as the main application.
- Database
    - Passed load test with high capacity provisioned than expected

Deployment Notes

The OPs team needs to run additional scripts/migrations when doing restart the application. Some extra resources also need to added to handle load in release window.

Deployment File
Release 2

Update the file for Release 2 to match the description in the scenario:

```
ApiVersion: apps/v1
kind: Deployment
metadata:
  name: app-deployment
  namespace: course4
  labels:
    app: mainApp
spec:
  Replicas: 6
  selector:
    matchLabels:
      app: mainApp
  template:
    metadata:
      labels:
        app: mainApp
    spec:
      containers:
      - name: mainApp
        image: nginx:latest
        resources:
          requests:
            memory: 320mb
            cpu: 250m
        ports:
        - containerPort: 80
      - name: order_processor
        image: nginx:latest
        resources:
          requests:
            memory: 1gb
            cpu: 1000m
        ports:
        - containerPort: 80
```

Scenario 2

On-Call Shift

On-Call Shift

Summary

Today is your first on-call shift as an SRE. During your shift, you will have to respond to alerts to keep the system running at its best using the on-call best practices learned in this course. During your on-call shift, make sure to be thinking of ways to reduce toil. After your on-call shift is over, you will be responsible for writing a summary of your shift and a post-mortem. On the following slides you will encounter several different “alerts” from your monitoring stack. Each “alert” will contain several different parts that will help you write your on-call log for your shift. Additionally, you’ll encounter an application outage that will require a post-mortem.

Alert Components

Summary -- This will be general knowledge about the systems involved that you would know if you had actually been working at the company. It will include a brief description of the systems involved as well information about how it is managed.

Standard Operating Procedure (SOP) -- This will be a short description of the steps to troubleshoot and potentially correct the cause of the alert.

Log and Monitoring Details -- This section will contain snippets of relevant logs and monitoring data (graphs, metrics, etc.) that are associated with responding to an alert.

On-call Log

After your on-call shift you’ll need to add to the on-call log. There is a provided sample template for you to use that includes all the necessary fields. Remember your on-call log is used to help track recurring alerts/issues as well as providing a record of the steps taken to resolve the issue.

Post-Mortem

Unfortunately there will be an application outage on your shift that will require a post-mortem. You will only be responsible for filling in your involvement, plus you’ll be in charge of creating an action plan and impact assessment.

On-Call Shift -- Alert 1

Order Processing Issues

Summary

You receive an alert that the number of Outstanding Orders is too high. Orders are processed by a separate component from the main application. It runs periodically (every hour currently) to batch process any open orders. Your team has set up some monitors to keep track of how well the order processor is doing.

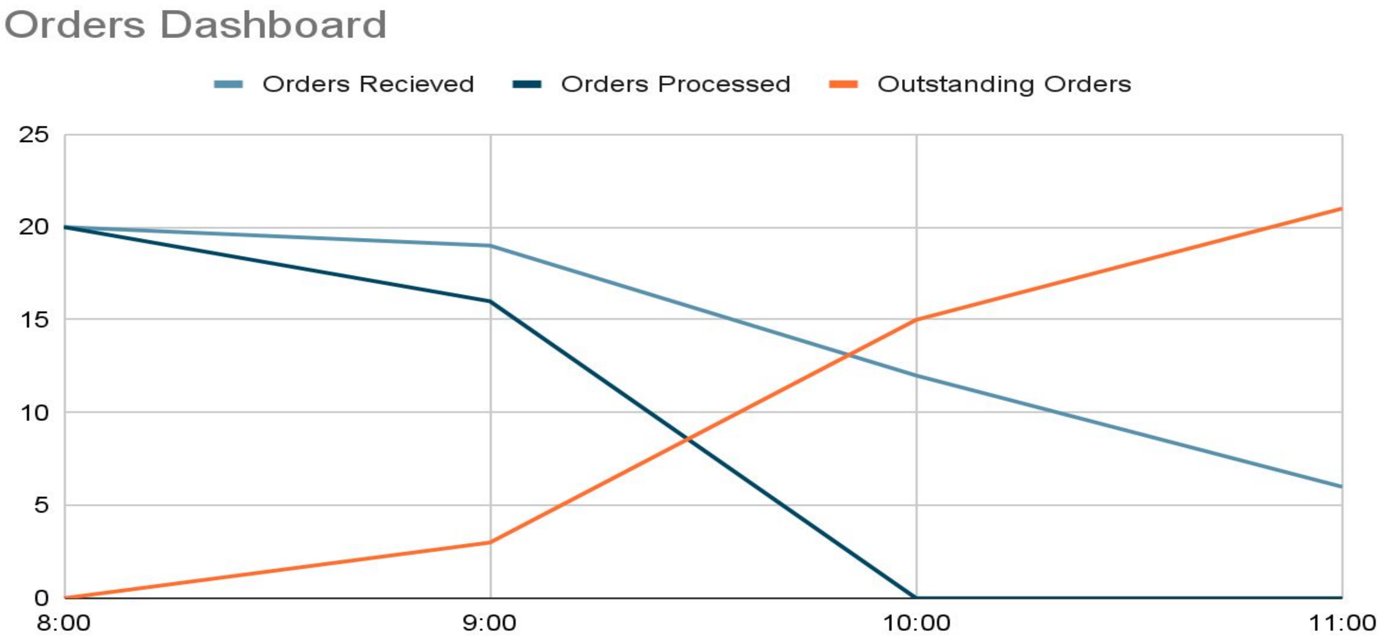
SOP

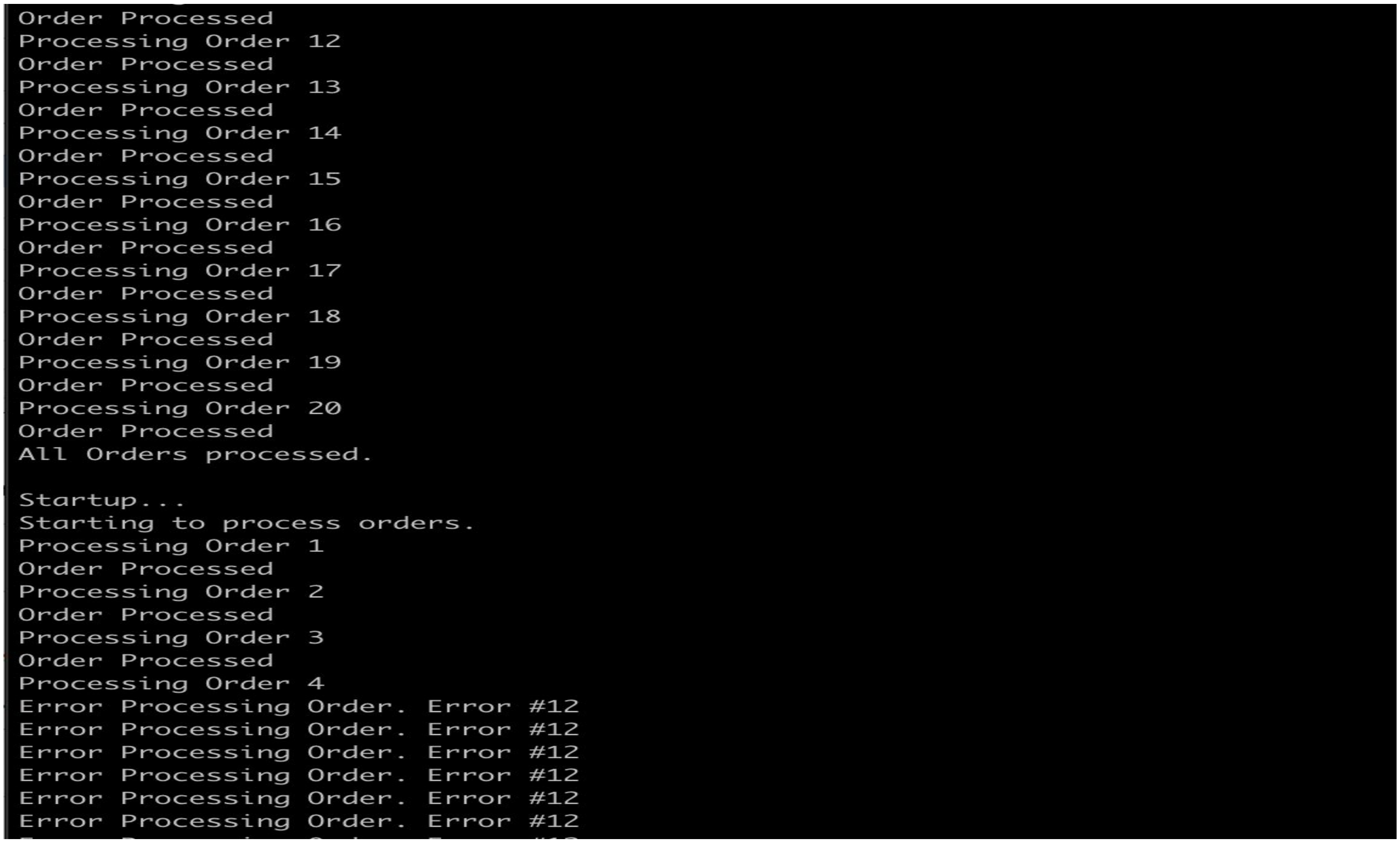
> Number of Outstanding Orders is Too High
>
> If this alert comes through you will need to check the dashboard to see if the Order Processor is overloaded with orders. If there is a high number of orders contact Ops to see if the processor should be run more frequently.
>
> There are logs at /home/sre/course4/order_processing.log. If the server is not overloaded, this is a good place to check for errors. If you encounter any errors, send a message to the developers so that they can troubleshoot.
>
> It is okay to restart this server during business hours. The Order Processor will pick up where it left off after a restart.

On-Call Shift -- Alert 1

Order Processing Issues, cont

Log/Monitoring Details





```
Order Processed
Processing Order 12
Order Processed
Processing Order 13
Order Processed
Processing Order 14
Order Processed
Processing Order 15
Order Processed
Processing Order 16
Order Processed
Processing Order 17
Order Processed
Processing Order 18
Order Processed
Processing Order 19
Order Processed
Processing Order 20
Order Processed
All Orders processed.

Startup...
Starting to process orders.
Processing Order 1
Order Processed
Processing Order 2
Order Processed
Processing Order 3
Order Processed
Processing Order 4
Error Processing Order. Error #12
Error Processing Order. Error #12
Error Processing Order. Error #12
Error Processing Order. Error #12
Error Processing Order. Error #12
Error Processing Order. Error #12
```

On-Call Shift -- Alert 2

Low Storage Alert

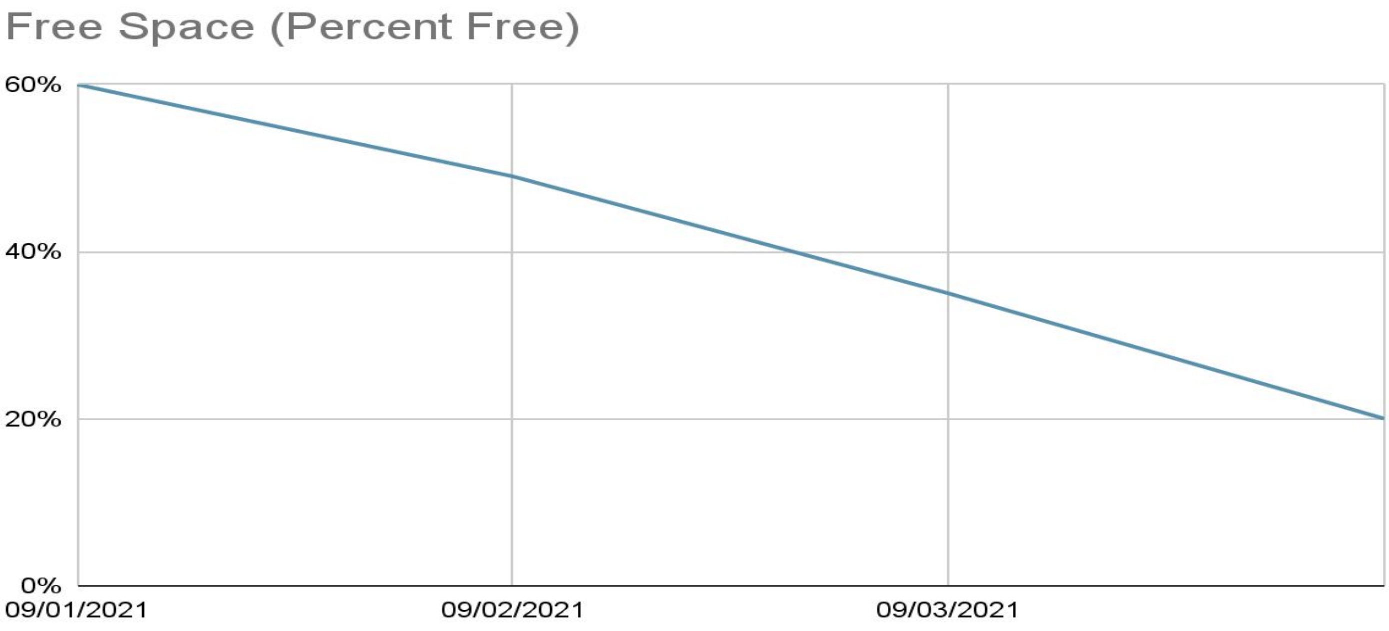
Summary

You receive an alert that the storage is running out on the mount where application logs are being written to. After consulting the SOP, you reach out to the team responsible for the server. They respond that Steve is normally in charge of handling the logs. Every morning he would run the commands listed in the run book, but he has been out sick for a week. The other members of the team forgot that it needed to be done, so the mount filled up.

SOP

> Low Storage
>
> Depending on the specific alert take the following action:
>
> /home/sre/course4/app.log -- If this mount is low on storage, reach out to Compliance. They will know what logs can be cleared out or will request additional storage.

Log



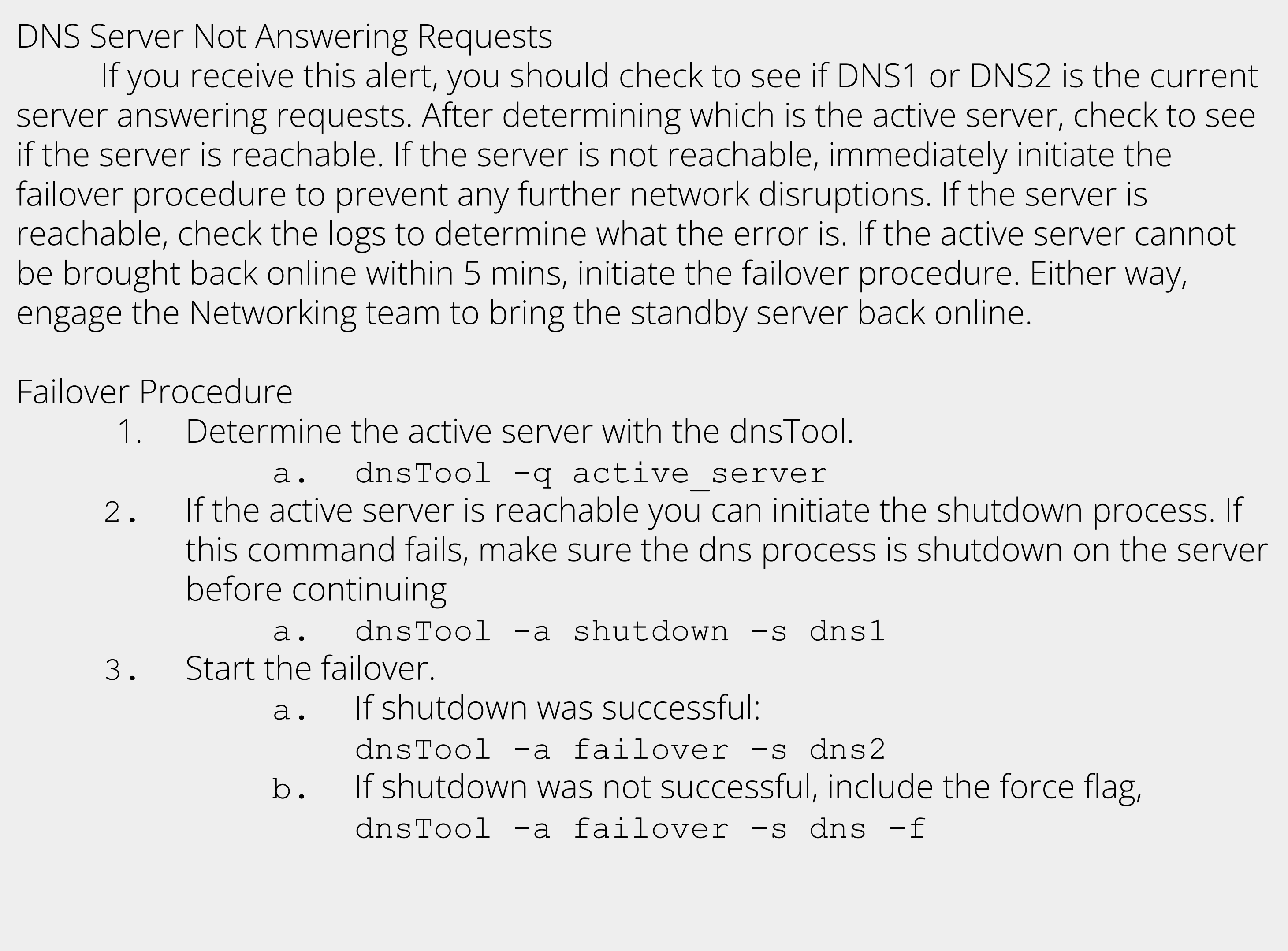
On-Call Shift -- Alert 3

DNS Troubles

Summary

The networking team recently added a secondary backup DNS server to increase reliability since the one they are using now tends to go down frequently. Your team has several checks in place monitoring the DNS servers to make sure they are up at all times.

SOP



DNS Server Not Answering Requests

If you receive this alert, you should check to see if DNS1 or DNS2 is the current server answering requests. After determining which is the active server, check to see if the server is reachable. If the server is not reachable, immediately initiate the failover procedure to prevent any further network disruptions. If the server is reachable, check the logs to determine what the error is. If the active server cannot be brought back online within 5 mins, initiate the failover procedure. Either way, engage the Networking team to bring the standby server back online.

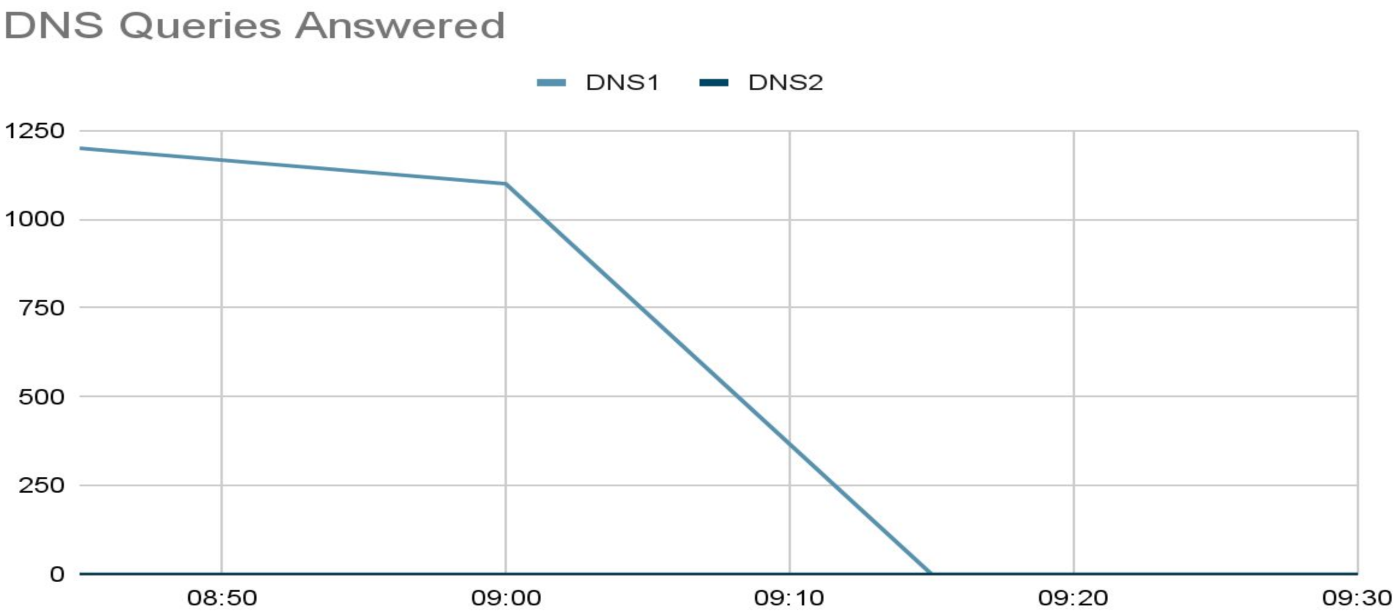
Failover Procedure

1. Determine the active server with the dnsTool.
   a. `dnsTool -q active_server`
2. If the active server is reachable you can initiate the shutdown process. If this command fails, make sure the dns process is shutdown on the server before continuing
   a. `dnsTool -a shutdown -s dns1`
3. Start the failover.
   a. If shutdown was successful:
      `dnsTool -a failover -s dns2`
   b. If shutdown was not successful, include the force flag,
      `dnsTool -a failover -s dns -f`

On-Call Shift -- Alert 3

DNS Troubles, cont

Log/Monitoring Details

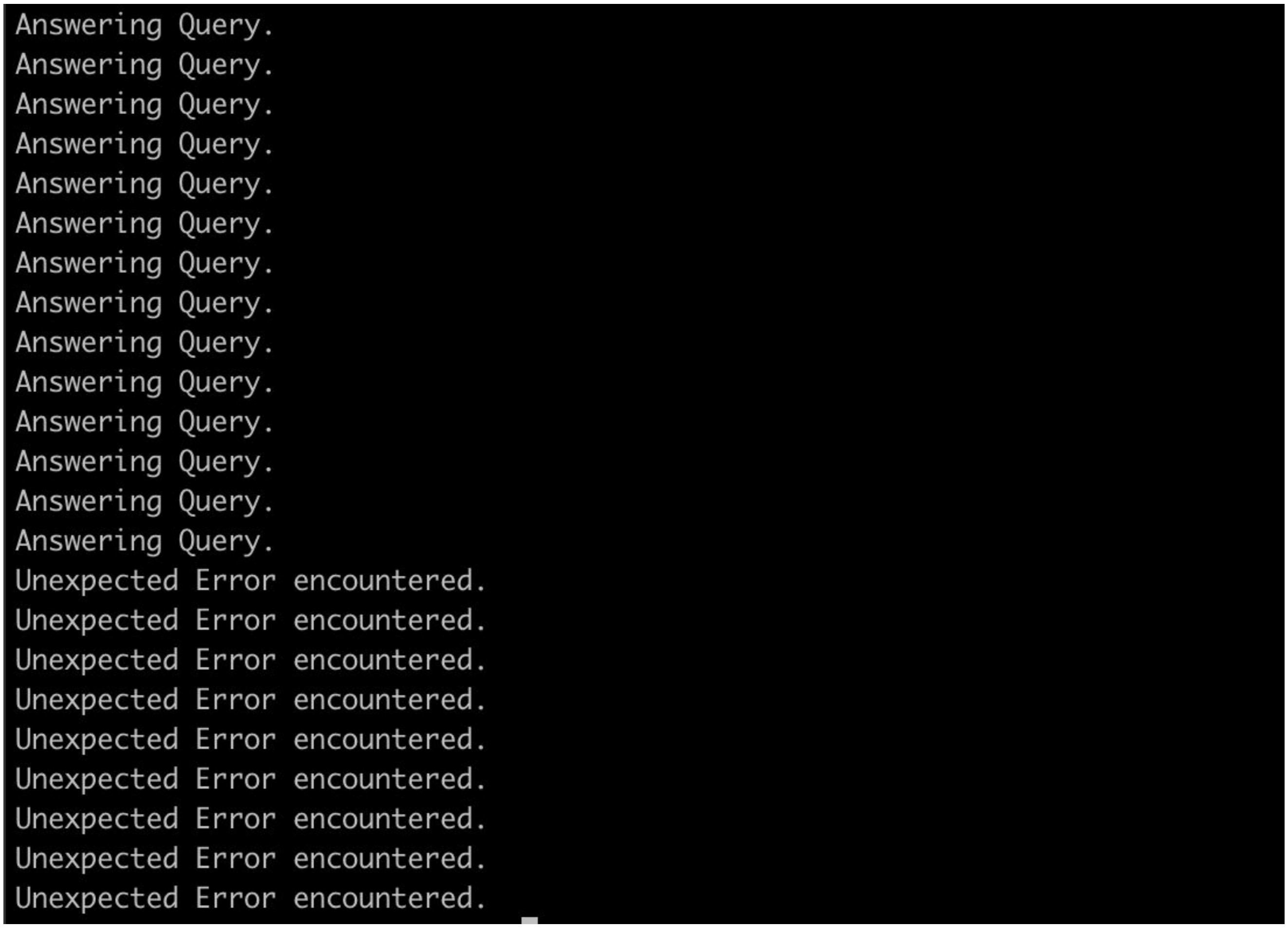


Networking Server Status Page	
Server	Status
DNS1	UP
DNS2	UP

On-Call Shift -- Alert 3

DNS Troubles, cont

Log/Monitoring Details



```
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Answering Query.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
Unexpected Error encountered.
```

On-Call Shift -- Alert 4

Application Outage

Summary

You receive the dreaded Application Down alert. Not only do you receive an alert for the application being down, but Customer Support also sent out a page to get all hands on deck for a report of the application being down.

SOP

> Application Down
>
> If you receive this alert, you need to act immediately. First, verify the application is indeed unreachable. If the application is unreachable, check to make sure the hosts are up and the application processes are running. You must start escalation for this immediately after verification the app is unreachable. Contact the following POCs:
>
> Customer Support -- Susan Vega
> Networking -- Bob Sparrow
> Ops -- Glen Hammer
> Database Admin -- Karen House
> Development Team – Gal Tree

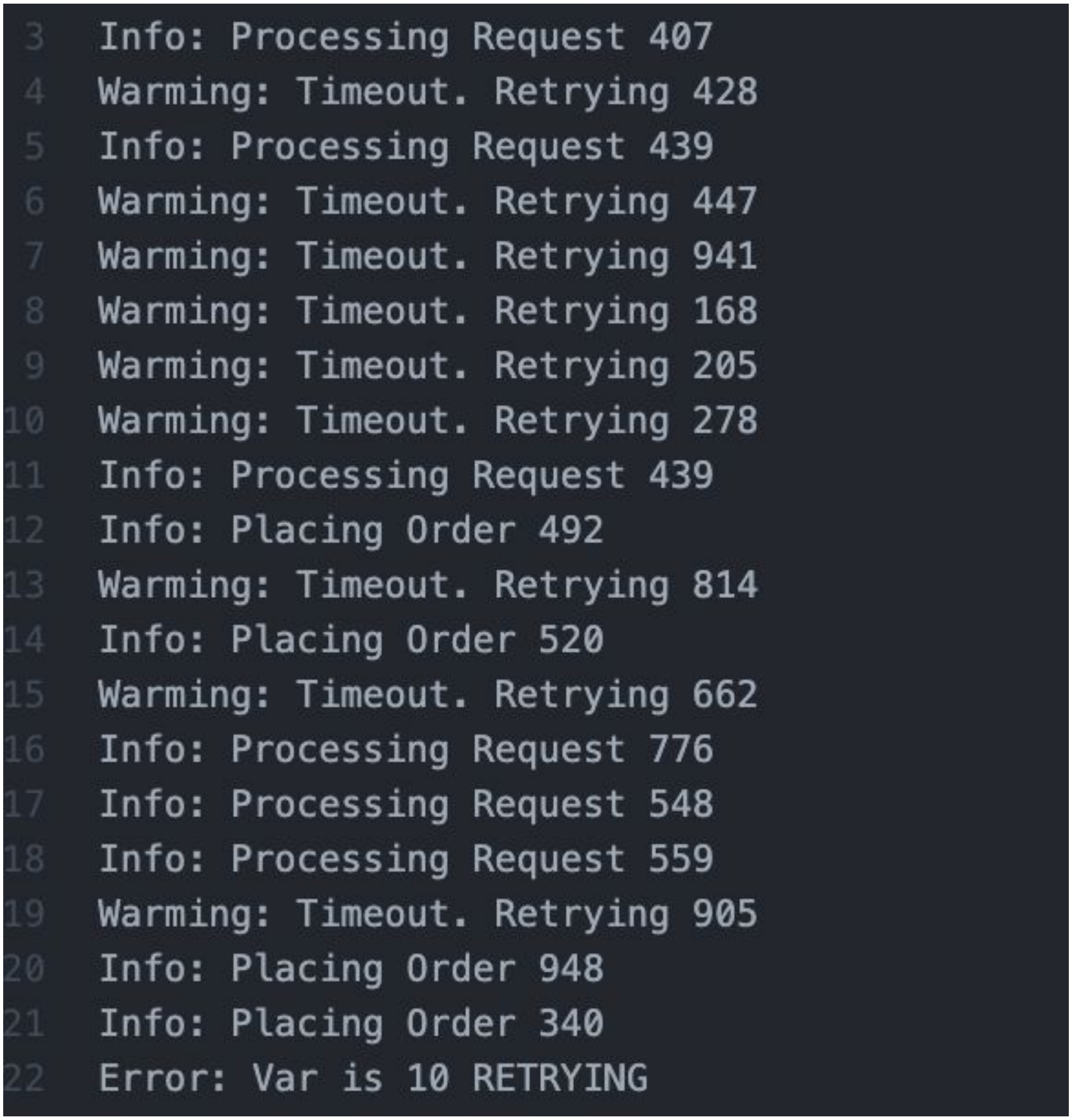
Lo

Main App Status	
Endpoint or Host	Status
exoticplant.plant	UNREACHABLE
planthost1.internal	UP
planthost2.internal	UP
exoticplant.plant.internal	UNREACHABLE

On-Call Shift -- Alert 4

Application Outage, cont

Log/Monitoring, cont.



```
Info: Processing Request 407
Warming: Timeout. Retrying 428
Info: Processing Request 439
Warming: Timeout. Retrying 447
Warming: Timeout. Retrying 941
Warming: Timeout. Retrying 168
Warming: Timeout. Retrying 205
Warming: Timeout. Retrying 278
Info: Processing Request 439
Info: Placing Order 492
Warming: Timeout. Retrying 814
Info: Placing Order 520
Warming: Timeout. Retrying 662
Info: Processing Request 776
Info: Processing Request 548
Info: Processing Request 559
Warming: Timeout. Retrying 905
Info: Placing Order 948
Info: Placing Order 340
Error: Var is 10 RETRYING
```

On-Call Shift -- Alert 4

Application Outage, cont

Log/Monitoring, cont.

09:15 Hey we have reports of an application outage and we can not reach the app either. **FROM: svega**

09:16 I have an alert for that too. I'm looking at things now, will start a communication channel to coordinate. Checking logs and app servers now. **FROM: YOU**

09:20 -- !svega !bsparrow !ghammer !khouse !gtree we have an application outage **FROM: YOU**.

0930 -- Everything looks good from the network **FROM: sparrow**

0932 -- I can access the DB and it is reporting back normal **FROM: khouse**

0935 -- Everything here looks normal. FROM: ghammer

0937 -- We are still reviewing logs and seeing if we can reproduce on our end FROM: gtree

0938 -- We should try restarting the app, Maybe that will help FROM: ghammer

0940 -- Maybe that will help. FROM: svega

0943 -- Okay I will try. Bringing down. FROM: YOU

0945 -- App is down. Bring back up. FROM: YOU

0947 -- App is starting. FROM: YOU

0952 -- Main app is back up. FROM: hammer

0955 -- App is still not respond. FROM: svega

0956 -- I'm sending you some new logs !gtree these look off FROM: hammer

1005 -- !sre !ghammer when was the last deploy? What were the details? This looks like a qa build. FROM: gtree

1007 -- I did a deploy with one of the devs to qa to do some testing. Let me check. FROM: ghammer

1010 -- I think there was a mixup when doing the deployment. The wrong scripts was used and that build was deployed to prod. FROM hammer

1011 -- Were there any migrations for that !ghammer FROM: khouse

1012 -- No, just code changes. FROM: hammer

1013 -- Thats good. We should be able to just revert back then. !svega

1015 -- Let me take down the app and redeploy it. FROM: YOU

1017 -- App is down. Bring back up. FROM: YOU

1023 -- App is starting. FROM: YOU

1026 -- Main app is back up. FROM: hammer

1030 -- Everything looks like it is responding now. FROM: svega

On-Call Summary Log

10:00 -- *Outstanding Orders is Too High*

Troubleshooting

- Checked the number of placed orders and it seems ok
- Checked the log at /home/sre/course4/order_processing.log and there was a an error with code type #12

Resolution

- Because it is safe to restart it, so forced the application to be restarted and communicate with developers for the fix if the issue is still there after restarting

On-Call Summary Log

09/04/2021 -- *Free Storage below 20%*

Troubleshooting

- Checked with Compliance team and get to know that some scripts does not get executed at the morning everyday

Resolution

- Requested Compliance team to clean up appropriate logs
- We can have this job to run everyday for free up storage

On-Call Summary Log

09:15 -- *DNS failure*

Troubleshooting

- By dashboard and cmd, it could get to know that DNS1 was the active one

- Both DNS was up at that time and checked the log, we can see that more than 5 minutes and it was still outage.

Resolution

- Started failover process and engaged network team to bring back the standby server online
- We could implement auto failover here to increase the high availability

On-Call Summary Log

09:15 -- *Application Outage*

Troubleshooting

- Checked the host was still UP
- Checked the application:
    - Planthost1, planhost2 was running
    - exoticplant was down
- Restarted once and nothing improved after first attempt

Resolution

- Due to critical situation, escalated to POCs
- Because of running wrong scripts, the QA build was sent to production. Fortunately, just code changes, so appropriate build has been deployed again on production.
- We should have CI/CD here to deploy on any configured environment

Post-Mortem Template

Application Outage -- 09:15

Stakeholders

Customer Support -- Susan Vega; Networking -- Bob Sparrow; Ops -- Glen Hammer; Database Admin -- Karen House; Development Team – Gal Tree

Incident Timeline

09:15 - Got alert for application outage happened (Vega and me)

09:30 - Checked the network and it was ok (Sparrow)

09:32 - Checked database server and it was ok (House)

09:43 - 1st attempt to restart the application (me)

09:55 - Application was back and still not responding (Vega)

10:10 - Figured out that wrong build was sent to production (Hammer)

10:15 - Redeployed appropriate build to production (me)

10:30 - Confirmed application function as expected (Vega)

Impact

The outage of the main application had a significant impact on the business, as it prevented customers from placing orders. This led to a loss of revenue for the company, as customers were unable to make purchases during the outage. The longer the outage lasted, the greater the potential revenue loss for the business. In addition to the immediate financial impact, the outage could also damage the reputation of the business if customers are dissatisfied with the level of service provided

Resolution

The incident was resolved by deploying the appropriate build to production, rather than the QA build. By using the correct build, the main application was able to function correctly and orders could be processed as expected. The QA build was identified as the root cause of the incident, and using the appropriate build helped to prevent similar issues from occurring in the future

Action Plan

To ensure that deployments are efficient and reliable, it is essential to have a Continuous Integration/Continuous Deployment (CI/CD) process in place. The CI/CD process should be automated and capable of deploying any build to any environment, including production. To minimize the risk of production issues, it is important to configure the process correctly and only allow builds that have passed QA to be deployed to production

Scenario 3

Toil Reduction

Toil Reduction Plan

Summary

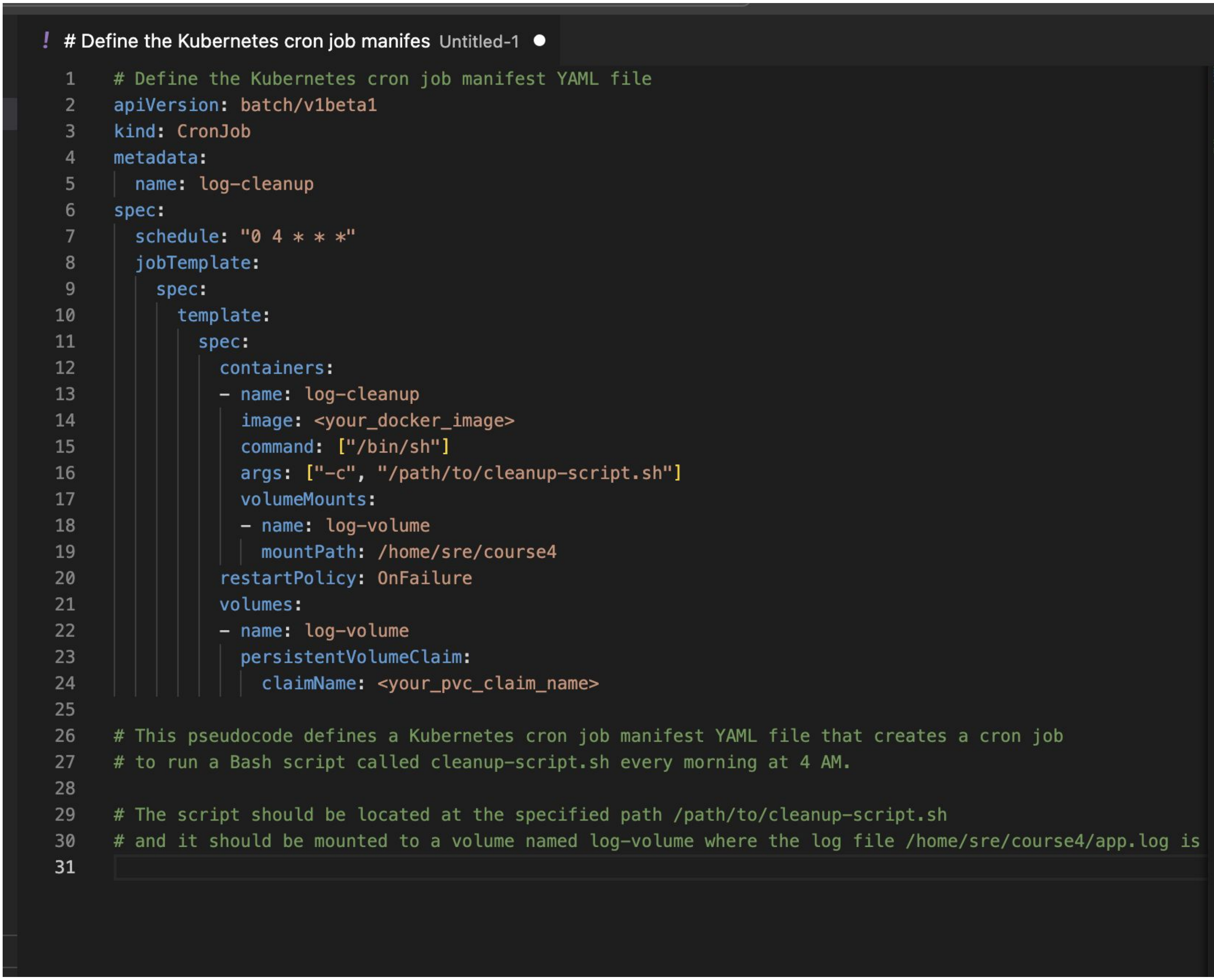
Now that you have spent some time on your own as an SRE, you now have to round out your week by handling some of the toil you encountered. Looking through the on-call summary, post-mortem, as-built design doc, and your experience, you decided that there are several ways to reduce toil. You start by listing out 5 of the major items for this week. For each one, you analyze the impact on the business and what you gain by automating the task. After that, you will need to implement three of these items in pseudocode to help your team move forward.

Current Toil Items

1. Manual task to clean up storage every morning
   - The impact of this manual task is that it consumes time and effort of the IT staff who have to perform it every day. If the task is not done correctly or on time, it could lead to storage space running out or slowing down the system. By automating this task, we can save time and improve the accuracy and consistency of the cleanup process, which can reduce the risk of storage-related issues.
2. Manual task to deploy build to environment
   - The impact of this manual task is that it can be time-consuming and error-prone, especially if there are multiple environments to deploy the build to. Manual deployments can result in configuration errors, downtime, or failed deployments. By automating the deployment process, we can reduce the risk of errors and improve the speed and consistency of the deployment process, which can lead to faster time-to-market for new features and improvements
3. Manual failover process
   - It can lead to significant downtime in the event of a failure, especially if the process is not well-documented or well-practiced. Manually initiating a failover can also lead to human errors, which can further prolong downtime. By automating the failover process, we can reduce the risk of downtime and improve the reliability and availability of the system.
4. Order processor is running all the time
   - If the order processor is running all the time, it can be a waste of resources and can potentially lead to unnecessary costs. Automatically adjusting the instance of the order processor to run hourly, rather than continuously, can help reduce resource usage and costs.
5. Manually run the script after start up the application
   - it can be time-consuming and can lead to errors if not done correctly. Manual execution can also result in inconsistencies in the behavior of the application, depending on who runs the script and when. By automating the script execution process, we can reduce the risk of errors and improve the consistency and reliability of the application.

Automation Implementation

Cron job to execute predefined scripts to clean up storage every morning



```yaml
# Define the Kubernetes cron job manifest YAML file
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: log-cleanup
spec:
  schedule: "0 4 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: log-cleanup
            image: <your_docker_image>
            command: ["/bin/sh"]
            args: ["-c", "/path/to/cleanup-script.sh"]
            volumeMounts:
            - name: log-volume
              mountPath: /home/sre/course4
          restartPolicy: OnFailure
          volumes:
          - name: log-volume
            persistentVolumeClaim:
              claimName: <your_pvc_claim_name>

# This pseudocode defines a Kubernetes cron job manifest YAML file that creates a cron job
# to run a Bash script called cleanup-script.sh every morning at 4 AM.

# The script should be located at the specified path /path/to/cleanup-script.sh
# and it should be mounted to a volume named log-volume where the log file /home/sre/course4/app.log is

```

Automation Implementation

Automate deployment

1. Create a deployment configuration file in YAML format to define the deployment settings for the production environment.
2. Set up a CI/CD pipeline with Github Actions to build the new changes and create a new container image with a unique tag for each build.
3. After the container image is built and tagged, use Kubernetes kubectl command-line tool to deploy the new container image to the production environment.
4. Use Kubernetes Rolling Update deployment strategy to deploy the new changes to production without any downtime.
5. Use Kubernetes Jobs or CronJobs to automate any additional tasks that need to be performed during deployment, such as running database migrations or updating configurations.
6. Use Kubernetes ConfigMaps or Secrets to store configuration data and secrets separately from the container image.
7. Set up a monitoring and alerting system to keep track of the deployment status and receive alerts if any issues arise during deployment.

Automation Implementation

Automatically failover with two DNS servers

1. Set up two DNS servers (primary and secondary) to handle domain name resolution
2. Create a Kubernetes service that exposes your application and assign it a DNS name
3. Configure the primary DNS server to point to the Kubernetes service
4. Configure the secondary DNS server to point to the primary DNS server
5. Set up a health check to monitor the availability of your application
6. Configure Kubernetes to automatically detect when the application is unavailable
7. When the health check detects that the application is down, Kubernetes will automatically spin up a new instance of the application on a healthy node
8. Kubernetes will update the DNS records to point to the new instance
9. The primary DNS server will update the secondary DNS server with the new IP address