15 FEBRUARY 2023

VINCENT PETIT, DUC-THOMAS NGUYEN, TIMOTHE PREVOT, ALEXANDRE PIEDVACHE, HUGO VILLEMADE

PROJECT REPORT

S TRATEGY ROADMAP

STRATEGY ROADMAP'S GOAL

This strategy roadmap has one main objective being the following: Achieve complete continuous deployment of the application.

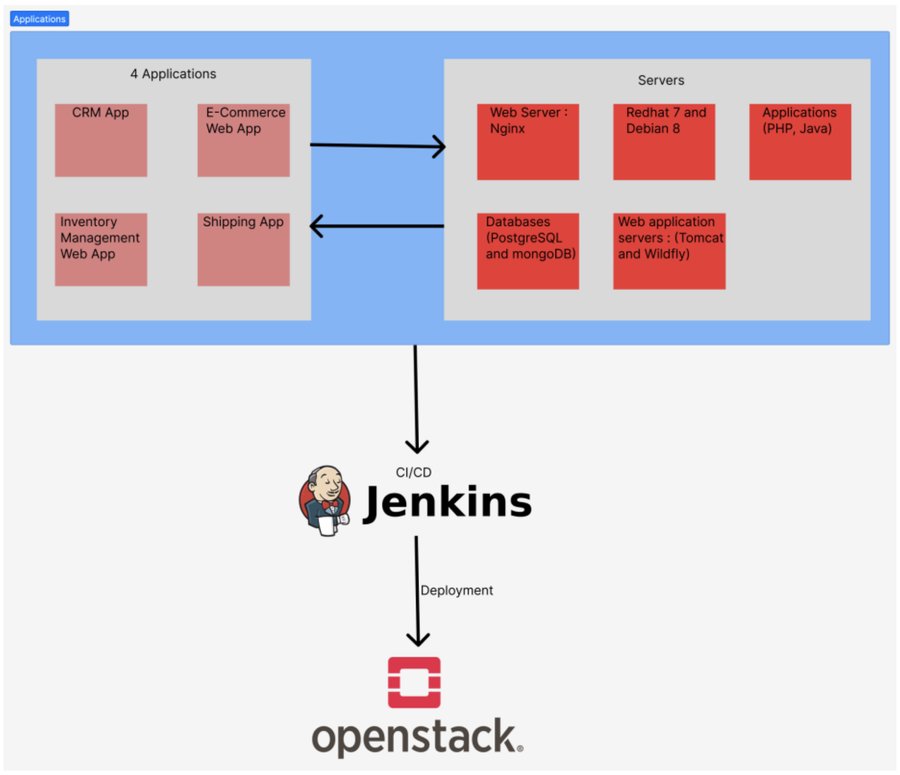
STAKEHOLDERS

Stakeholders are those participating in the project (from developers to business leaders). In our case the entire IT team is part of the project.

First, we have the designers, architects and managers who help conceive the roadmap. Then we have employees dealing with the technical parts like implementations of the roadmap. They are composed of developers, testers, integrators…

We find it important to have everyone on board with such a project. That allows us to ensure a successful outcome.

BLUEPRINT OF EXISTING SYSTEM



1. Environments

The following environments are the ones used by the company: development, integration, acceptance, preproduction, production. This represents a typical set of envs.

2. Resources

We have at our disposal a complete infrastructure composed of 16 servers. The detailed infrastructure is as followed:

- <u>Web servers</u>
  They are running web servers with Nginx.

- Web application servers
  They are using Tomcat and Wildfly altogether.

- Databases
  The databases being used are PostgreSQL and MongoDB.

- Application development
  Languages of choice for them are PHP and Java for app development.

- Operating systems
  Their servers are running versions of Redhat 7 and Debian 8. They are UNIX based.

- Deployment model
  They deploy their applications on VM on a private cloud built on top of Openstack.

3. Process

Existing systems rely on standard processes described below:

- Inputs are user requests.

- The processing is done by the web servers which handles request and transmit them to their different application servers. After reception they process the requests and access data.

- Outputs are responses sent to the user requesting data.

4. Locations

The company has an IT company divided in 2 parts:

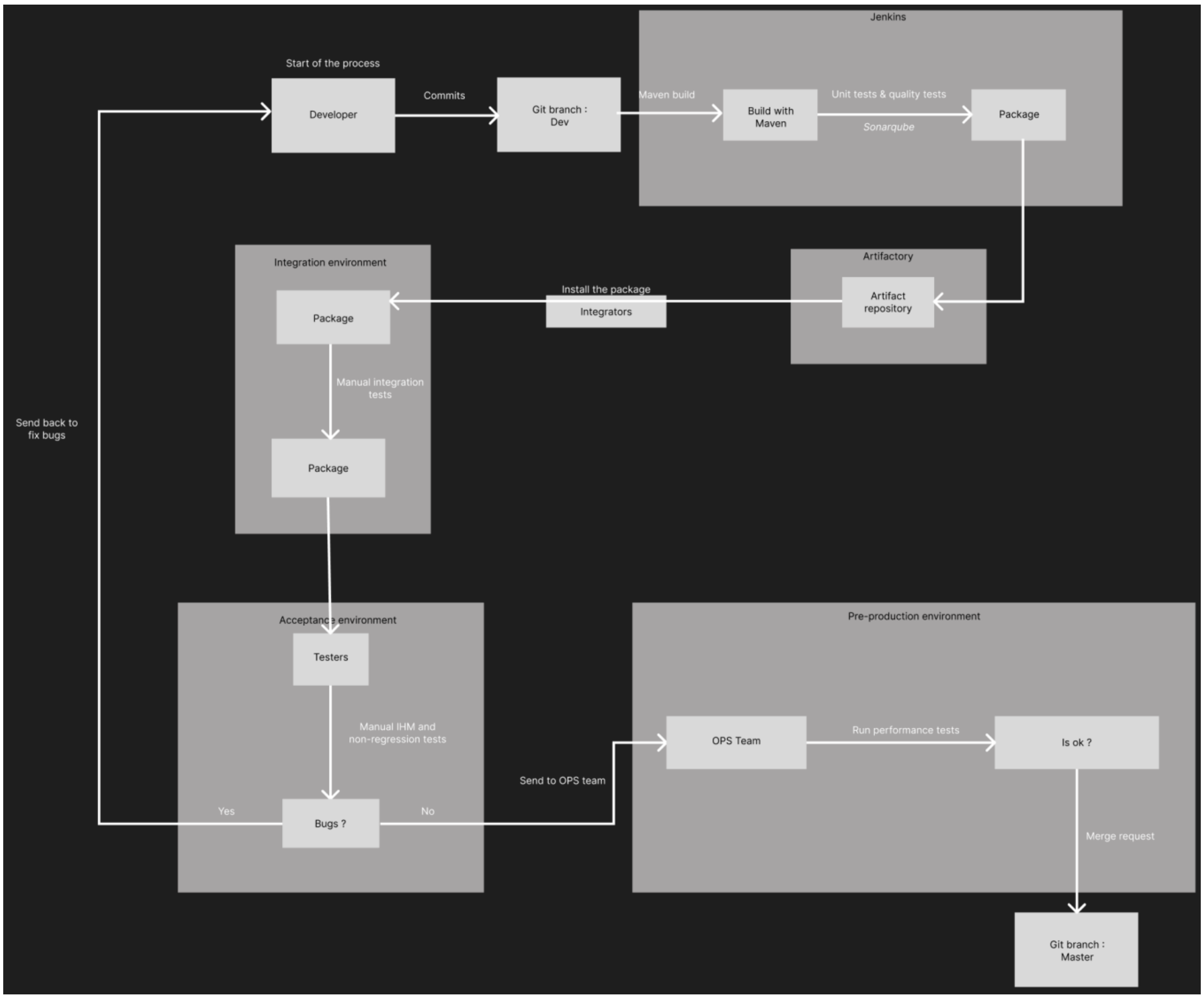
- Developers in Geneva

- Rest of IT team in Paris

5. CI/CD Toolchain

- The build tool used is Maven.
- The source code management tool of choice is GitHub.
- Integration servers are using Jenkins.
- For unit testing they use JUnit and phpUnit.
- One aspect related to unit testing is code quality tests for which they utilize Sonarqube.
- For the last steps they use Artifactory to publish artifacts.
- With those artifacts they deploy their solutions via scripts in Bash and Python.

6. Delivery Process



- GitHub repositories have two long-lived branches called DEV (development) and MAIN (production).

- Commits are meant to be pushed on the DEV branch only.

- Jenkins builds features with Maven followed by unit & code quality testing with Sonarqube. If tests pass artifacts are published.

- Integrators install the package on integration servers by following guidelines then deploy on acceptance environment.
- Testers run manual IHM tests and non-regression tests. If passed continue to next step otherwise send a report.
- Operations teams install the given package in preproduction and test the performance. If satisfied they open a PR to merge with MAIN.
- As a final step the release manager approves the PR and proceed to merging the branch.

By following the process described above they can port their solutions into production.

BLUEPRINT OF TARGET SOLUTION

Here is the link of our github organization: https://github.com/ecoshop-devops-m2

The main goals of our target solution are to be faster than the current system (which takes at least 3 weeks), and to be less dependent on human actions, so this means more automation. We also want our solution to be safer than the current one, so we'll use some monitoring tools to help us mitigate problems.

Technology stack:

- Jenkins pipeline for the build
- GitHub Actions pipeline for testing
- Server to handle the monitoring with a docker-compose of:
    a. Cadvisor: for data related to the health of our containers.
    b. NodeExporter: for data related to the health of our host server.
    c. AlertManager: to send Slack notifications in case of a Prometheus alert.

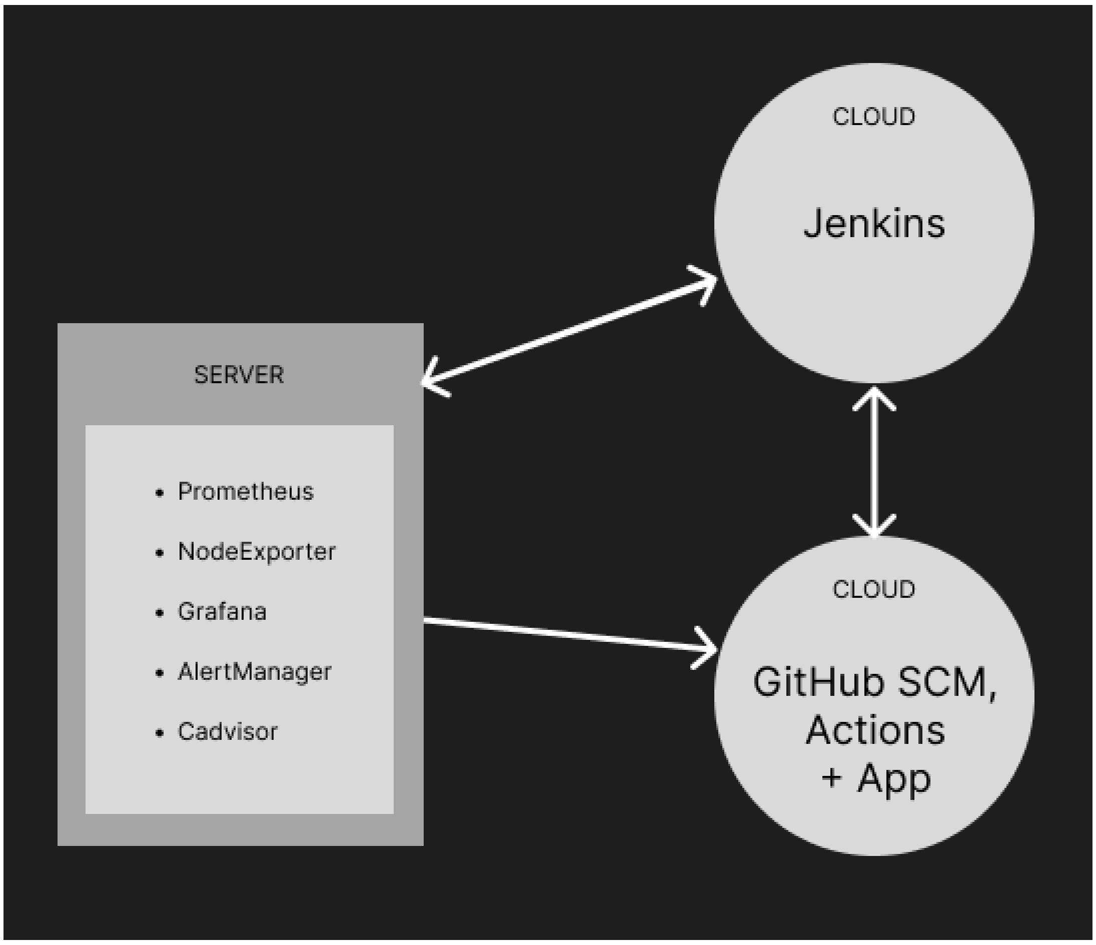
d. Grafana: dashboard regrouping all our data.

e. Prometheus: For endpoint scrapping, it fetches the data from the containers and serves it to Grafana.

The infrastructure consists of a Jenkins pipeline hosted on a VPS, a GitHub Actions pipeline hosted on GitHub, and a server with a Docker container hosting our monitoring tools.

PROTOTYPE'S ARCHITECTURE (TBD)

1. Document the solution: Finally, document the architecture of your target solution in detail. This will include the system architecture diagram, a description of each component and how it works, and any other relevant information. This documentation will be valuable for future reference and for others who need to understand how the system works.Architecture of prototype`



For our prototype, we separated our architecture in three parts:
- 2 cloud (1 VPS and 1 external service)
- 1 VPS or on-site server

We host the code of the application and of the monitoring + CI/CD configuration files on GitHub as two separate repositories.
The application repo has a main protected branch on which you cannot directly push except if you are an administrator. To add new features, you must code on a separate branch dedicated to this feature development.
In order to merge the dev branch, a GitHub Action will run all the tests from the application. Merge will only be allowed if all tests pass, meaning that if a single test fails, the main branch will not be polluted with bugged code.
After the merge, another GitHub Action will run to ask the Jenkins server to run the build pipeline.

The Jenkins instance can be deployed on a VPS and will have a pipeline dedicated to build the application after a merge to the main branch and send the artifact to

Artifactory. In our POC, the pipeline is only valid as a skeleton but was not necessary to the MVP, meaning that it does not perform any meaningful action. The pipeline is stored on the devops repo in CI/CD folder in the Jenkinsfile.

The Jenkins instance's metrics are collected by Prometheus for the Monitoring Stack.

The Monitoring Stack is a suite of tools dedicated to ensuring the good operation of the DevOps lifecycle. All config, alerting rules and dashboards of these tools are in the monitoring folder of the devops repo. Each tool has its own purpose:

Prometheus scraps data from the Jenkins instance (health, build status) and from the other monitoring services and the host of this stack. It is also responsible for the alerting rules in case of a problem on the DevOps stack.

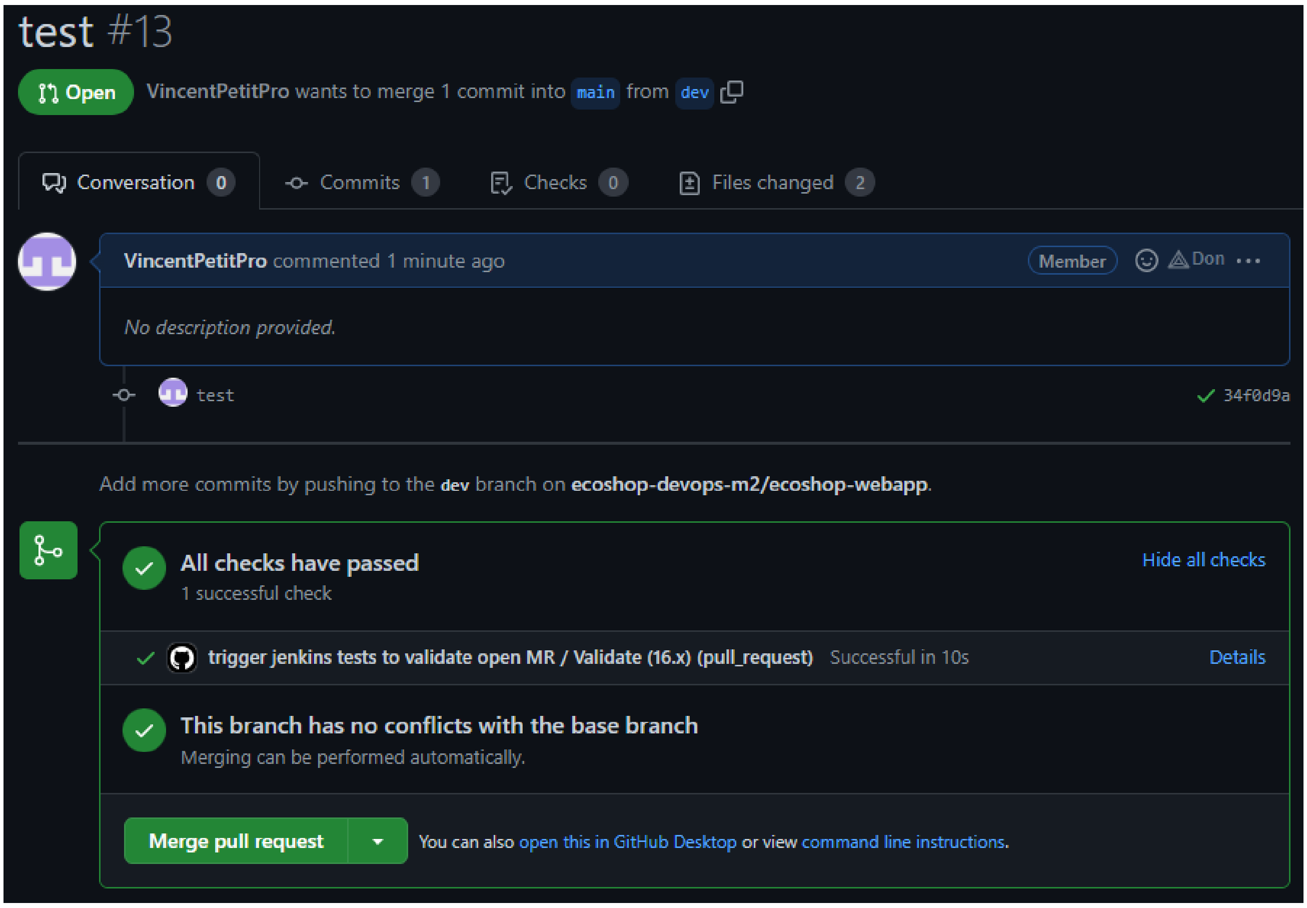
Grafana is used as dashboards to visualize the data that Prometheus scrapped. There are 3 distinct dashboard, each for a different part of the toolchain. 1 is for monitoring the services, 1 for the services health and the last one for the host health.

Alertmanager is working with the alerting rules defined in Prometheus, it will send a Slack message on a specific channel each time that a rule has fired.

Nodeexporter and Cadvisor are data exporter used to monitor containers and host health.

CLEAR AND REUSABLE TOOLCHAIN (TBD)

We start by modifying the code on the dev branch of the app. Then we push the code and create a pull request. Once the pull request has been created, the GitHyb Action runner will run tests to prevent merging bad code. The merge will only be possible if all tests pass.


test #13

Open VincentPetitPro wants to merge 1 commit into `main` from `dev`

Conversation 0 · Commits 1 · Checks 0 · Files changed 2

VincentPetitPro commented 1 minute ago · Member

No description provided.

test · 34f0d9a

Add more commits by pushing to the **dev** branch on **ecoshop-devops-m2/ecoshop-webapp**.

All checks have passed
1 successful check
Hide all checks

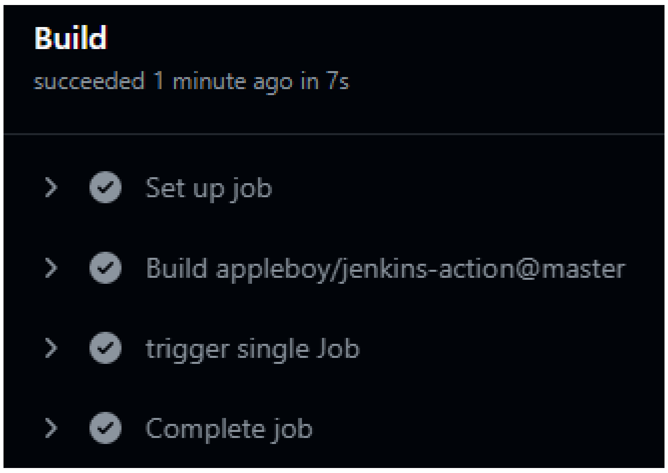
trigger jenkins tests to validate open MR / Validate (16.x) (pull_request) Successful in 10s · Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request · You can also open this in GitHub Desktop or view command line instructions.

Figure 1 All tests passed, the job is validated, and we can merge this PR.

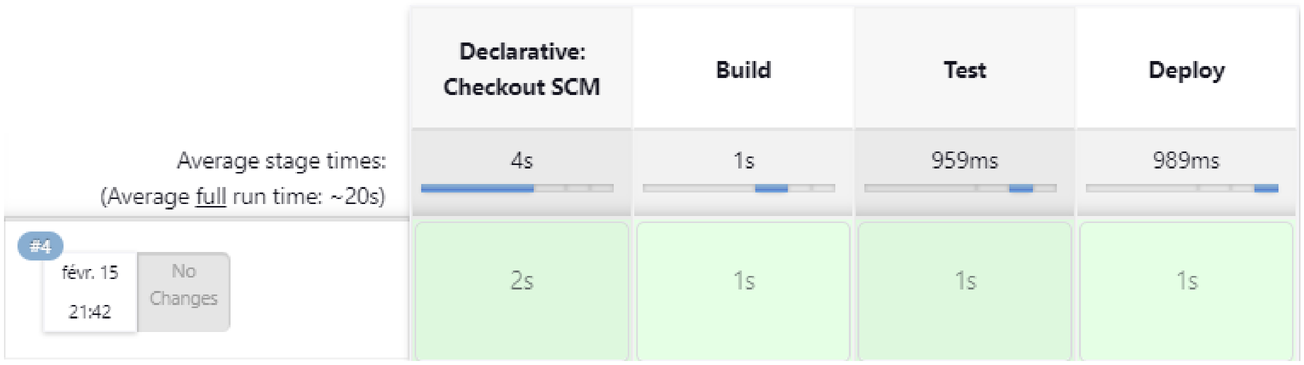
Once merged, another GitHub Actions runner will ask our Jenkins to run the pipeline via a POST request to the /job/JOB_NAME/build endpoint.



Build
succeeded 1 minute ago in 7s

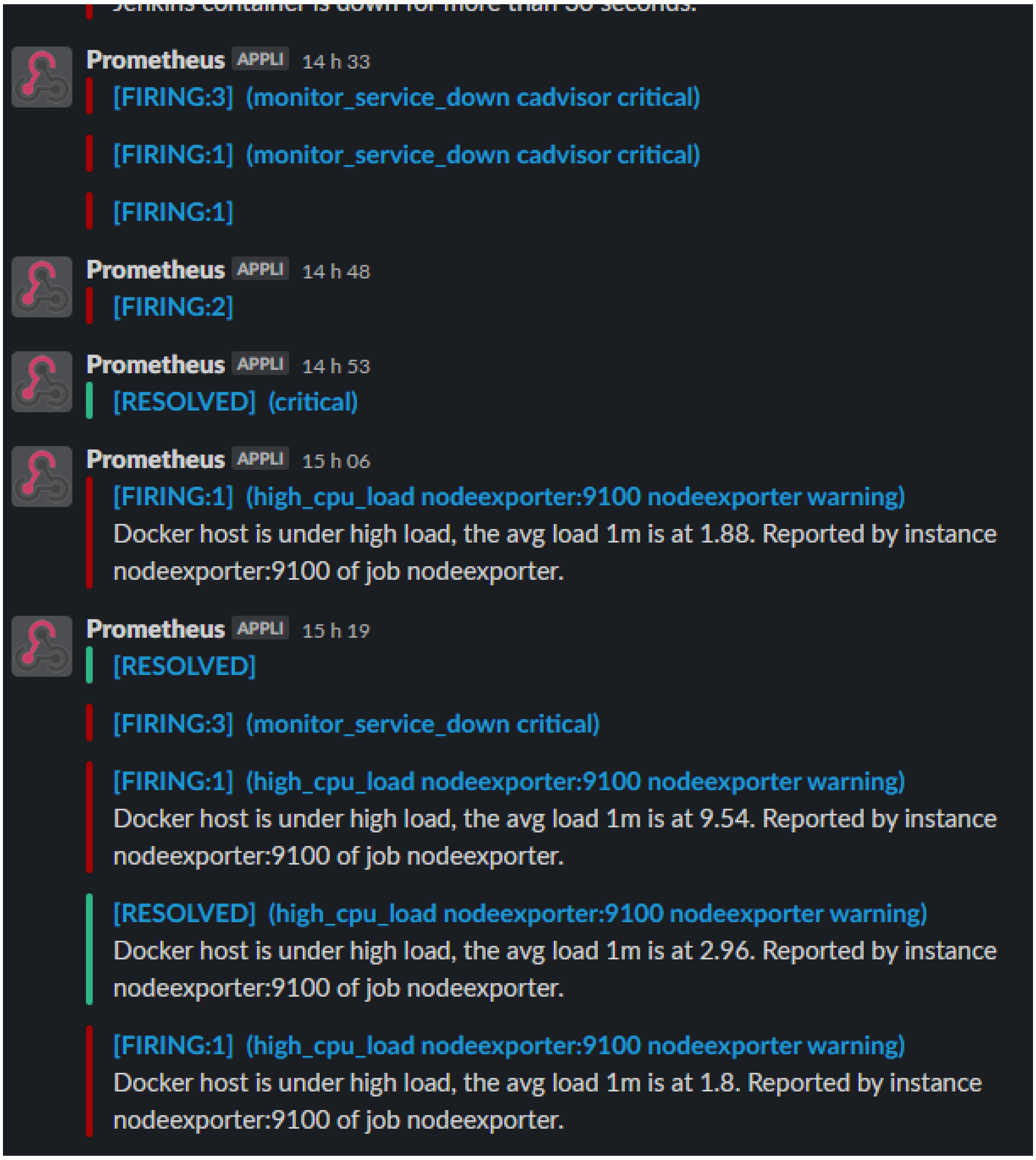
- Set up job
- Build appleboy/jenkins-action@master
- trigger single Job
- Complete job

Then the pipeline builds the artifact and post it on artifactory.



	Declarative: Checkout SCM	Build	Test	Deploy
Average stage times: (Average full run time: ~20s)	4s	1s	959ms	989ms
#4 févr. 15 21:42 No Changes	2s	1s	1s	1s

If anything happens to the toolchain, alertmanager will tell you accordingly to the alerting rules of Prometheus via Slack messages.



```
Jenkins container is down for more than 30 seconds.

Prometheus APPLI 14 h 33
[FIRING:3] (monitor_service_down cadvisor critical)

[FIRING:1] (monitor_service_down cadvisor critical)

[FIRING:1]

Prometheus APPLI 14 h 48
[FIRING:2]

Prometheus APPLI 14 h 53
[RESOLVED] (critical)

Prometheus APPLI 15 h 06
[FIRING:1] (high_cpu_load nodeexporter:9100 nodeexporter warning)
Docker host is under high load, the avg load 1m is at 1.88. Reported by instance
nodeexporter:9100 of job nodeexporter.

Prometheus APPLI 15 h 19
[RESOLVED]

[FIRING:3] (monitor_service_down critical)

[FIRING:1] (high_cpu_load nodeexporter:9100 nodeexporter warning)
Docker host is under high load, the avg load 1m is at 9.54. Reported by instance
nodeexporter:9100 of job nodeexporter.

[RESOLVED] (high_cpu_load nodeexporter:9100 nodeexporter warning)
Docker host is under high load, the avg load 1m is at 2.96. Reported by instance
nodeexporter:9100 of job nodeexporter.

[FIRING:1] (high_cpu_load nodeexporter:9100 nodeexporter warning)
Docker host is under high load, the avg load 1m is at 1.8. Reported by instance
nodeexporter:9100 of job nodeexporter.
```

CONTINUOUS DEPLOYMENT

Continuous Deployment is an approach to software development that automates the building, testing, and deployment of code changes to production environments without the need for human intervention. This method enables teams to swiftly and effectively introduce new features and address bugs for their users. On the other hand, Continuous Delivery follows a similar process but necessitates manual approval before the changes are released into production. This approach allows teams to assess changes in a staging environment prior to their release to production.

Some recommended practices for creating a Continuous Deployment pipeline are:

- Automating the process of building, testing, and deploying software updates
- Setting up automatic rollback procedures to address any issues that arise during deployment
- Regularly performing performance and load testing to ensure the system's stability and scalability
- Establishing a robust configuration management and version control system
- Implementing ongoing monitoring and logging of the deployed systems to facilitate rapid identification and resolution of issues.

The deployment phase of application development often employs a range of tools, such as:

- Jenkins
- Travis CI
- CircleCI
- AWS CodeDeploy
- Heroku
- Google Cloud Build
- GitLab CI/CD

- Azure DevOps.

CONTAINERIZATION

When applications are moved to production, containerization offers various advantages, such as:

- Enhancing the portability and consistency of application environments across different stages of development, testing, and production.

- Improving the utilization of resources compared to traditional virtualization.

- Enabling faster deployment and scaling of applications.

- Enhancing security by isolating applications and their dependencies.

To ensure the successful use of containers in production, best practices should be followed, including:

- Creating and maintaining secure base images.

- Automating the build and deployment process with a CI/CD pipeline.

- Monitoring the performance and resource usage of containers.

- Implementing a centralized logging solution.

- Regularly updating and patching containers and their dependencies.

To scale and orchestrate containers, tools such as Kubernetes, Docker Swarm, Apache Mesos, Amazon ECS, and OpenShift can be utilized. These tools offer features such as automatic scaling, rolling updates, service discovery, and load balancing of containers.

It is important to note that successful containerization requires a comprehensive approach that includes the use of appropriate tools and best practices. In addition, effective management and governance are necessary to ensure the smooth operation of containerized applications in production.

DEPLOYMENT STRATEGIES

In addition to careful planning and testing, there are some additional considerations and complexities to keep in mind when implementing these deployment strategies.

For example, in Blue-Green Deployment, it is important to ensure that the "green" environment is identical to the "blue" environment in every way, including configuration, software versions, and underlying infrastructure. Any differences between the two environments could result in unexpected behavior during deployment and could potentially cause downtime.

Similarly, in Rolling Deployment, it is important to carefully choose the subset of servers that will be updated first. This subset should be small enough to limit the impact on the overall system if issues arise, but large enough to provide meaningful results and allow for thorough testing.

Another important consideration in both deployment strategies is data consistency. If data is being updated or migrated during the deployment process it is essential to ensure that the data is consistent across all servers and environments. If you can't ensure that then your results could be data loss or corruption. This would end-up to serious consequences for your users and your applications.

Both solutions require careful planning but when done properly Blue-Green and Rolling Deployment can be effective in reducing downtime and improving the user experience. By taking these considerations into account and implementing these strategies correctly, organizations can improve the reliability and availability of their applications. Ultimately all of that would provide a better experience for your users.

Both of these strategies can increase the reliability of deployments by minimizing the risk of downtime and enabling faster recovery in the event of a failure. Moreover they can provide more opportunities for testing and validation, leading to a higher quality release.

It is worth noting that there are other deployment strategies as well, such as Canary and A/B testing. These strategies involve deploying changes to a subset of users or traffic to validate the changes before rolling them out to the entire user base. These deployment strategies can also help improve the reliability and quality of deployments.

MONITORING

These are some of the commonly used monitoring tools for quickly detecting service downtime: Nagios, Zabbix, Datadog, NewRelic, Prometheus…

To integrate a monitoring tool into your system and ensure consistent monitoring across all environments, consider the following steps:

Choose a monitoring tool that fits your infrastructure and requirements.

Configure the tool to monitor the desired metrics and services.

Set up alerts to notify your team when a service goes down or when performance degrades.

Integrate the monitoring tool into your CI/CD pipeline.

Implementing a monitoring tool can provide several benefits such as improved uptime and reliability of services, early detection and resolution of issues, better visibility into the performance and behavior of services, and improved efficiency in detecting and resolving issues.

To effectively utilize logs, follow these steps:

Centralize log collection and storage to simplify log management and analysis.

Implement log parsing and indexing to easily search and analyze logs.

Set up alerts and notifications based on log data to quickly detect and address issues.

Integrate log analysis tools to gain insights into application behavior and performance.

Implement log retention policies to manage storage costs and comply with regulatory requirements.

By implementing these steps, you can gain better insights into the behavior and performance of your applications, detect and address issues early, and improve the overall reliability and efficiency of your system.

TESTS

When testing your projects end-to-end that means every aspects of your solutions. Therefore you have to use different kind of tools to achieve a complete testing suite.
The first one would be test automation frameworks. The most popular ones are Selenium, Cypress, Puppeteer or Appium.

To run automated tests you have to utilize CI/CD tools that are a must have in our case. Moreover this is what allows us to build, run and deploy. In order to run our tests available solutions are Jenkins, Travis CI and Circle CI.

End-to-end testing also means testing what your application is using, for instance a custom API. To test your APIs you can use Postman and RestAssured to test your endpoints and their corresponding responses.

If you want to run performance tests (replicate heavy traffic, stress on your apps) they are tools like JMeter to measure those metrics.

(If you want to run tests on different environments you have to use VMs. You can use Docker or Kubernetes to manage your VMs.)

Best practices we can give are the following:

- Start testing ASAP in your project lifecycle. This can help catch bugs early in your development process. The sooner you find a defect, the easier it will be to fix it.

- Have a consistent testing strategy across all your different environments. A given defect should happen in every environment.

- In your testing strategy, use many types of testing: unit testing, integration testing, end-to-end testing etc and automate them.

- Make sure your tests run at every step of your code's modifications meaning after commits are pushed (including open PRs and merge commits).

- If you modify your codebase, make sure the related tests are still working correctly. If not fix them and DO NOT keep wrong tests.

- The scope of a single test should only apply to one use case/feature, allowing you to write understandable tests and run them in parallel to prevent useless dependencies.

- Everyone must participate in testing in order to have as much feedback as possible. Their requirements are probably not the same as yours.

- Stay up-to-date with your codebase.

RELIABILITY

SRE stands for Site Reliability Engineering. It is an approach first used at Google which consists of managing and maintaining highly reliable IT systems.
the goal of this approach is to use a combination of software engineering practices and operation management to produce software systems that are both scalable and reliable. SRE is focused towards building, operating and maintaining large-scale distributed systems.

SRE is related to DevOps as it encapsulates DevOps methods and manages it. SRE is focused on the operations side of the process and insists on the reliability of the systems. You can see one as management oriented and the other as technical oriented.

In order to implement SRE there are multiple steps to follow:

- You must define the Service Level Objectives or SLO. They represents measures of your system's levels of service compared to an expected metric of how much your system should serve those services. This metric is used to get an idea of how your system is performing and gives you reliability level.

- SRE also insists on the use of automation in order to reduce human errors and to speed up the delivery process of your different solutions. You can achieve this by automating deployment and monitoring tasks to get the needed metrics used for your system's reliability evaluation.

- About monitoring you have to keep an eye on your solution's performance, availability and all the other important metrics as it is critical to have a solid knowledge of how your systems are performing.

- You have to establish a structured incident response process to learn and improve your process. This documentation will prevent similar events from happening again.

- SRE is all about collaboration among your teams so make sure to create a suitable environment of work that helps towards collaborative work.

CI/CD FROM OUR EXPERIENCES

Git strategy and good practices

As mentioned in the guidelines two long-living branches are used in Git. These are main and dev. But that does not mean you only have to use only two branches. Temporary branches are a must and have to be used in order to keep your git history clean (one branch per feature). Moreover workflows and tags can be used to improve the development experience.

Here is a a git strategy that we recommend using branch per feature, 2 tags, 5 environments called dev, integration, acceptance, pre-production and production.

1. Push on dev branch ==> deploys to dev env
   — pushing to dev branch implies you merge your feature branch into dev

2. A PR from dev to main ==> deploys to integration env

3. Push on main branch ==> deploys to acceptance env

4. Add a tag on main ==> deploys to pre-production env

5. Manual workflow on a tag ==> deploys to production env
   — if you want a fully automated process you could have an automatic trigger that adds a the tag on tis own

Limits of continuous deployment

We happened to talk with companies that refuse to implement a fully automated pipeline as they are reticent to human errors. They are many constraints like downtime, weekends and deployment failure that big companies want to avoid absolutely. To prevent that they have manual workflows they trigger after careful checking. Some have automated workflows that waits for for certain conditions to proceed like human validation, time conditions (for instance only deploy before 10am).