**CHAPTER 4**

**SYSTEM ANALYSIS**

**4.1 PROJECT DESCRIPTION**

Today in IT sector there is a very tough competition with each and everything. None of the company wants to lag behind with any respect. Say its time, cost or effort. Since 2 years back all the companies were working on cloud with tools by installations and configurations. There were 4 big companies who were working on cutting edge technologies of cloud and automation. They say it SRE(Site reliability Engineering).

Our mission is to automate all the manual installations for the release and deployment of the software. We will be using various tools and software like Git, jfrog, Jenkins, jira, maven, git etc. these almost all of them work with each other. In the end we will be using Puppet to automate all the tools. CICD is a process for continuous development, testing, and delivery of new code.

This system automate all the manual installations of the tools for the release. Automate tests that run fast and have good coverage on code. Automate the deployment of the software.

**4.1.2 CONTINUOUS INTEGRATION**

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily. In this project we are going to integrate continuously in a Agile methodology.

Continuous integration in practice, it is a use of a version control tool (CVS, SVN, Git, etc.). An automated build and product release process. Instrumentation of the build process to trigger unit and acceptance tests ”every time any change is published to version control”. In the event of even a single test failing, alerting the team of a ”broken build” so that the team can reach a stable, releasable baseline again soonest.

Optionally, the use of a tool such as a continuous integration server, which automates the process of integration, testing and reporting of test results.

The team responds to a broken build, suggesting that a defect may have been detected. If the team is aware of defects, but tolerates them or continues working on a product that isn’t in a releasable state, the term continuous integration no longer applies, irrespective of tooling.

In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. Grady Booch first named and proposed CI in his 1991 method, although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day - perhaps as many as tens of times per day.

**4.1.3 CONTINUOUS TESTING**

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate. Here we are using the methodology continuous testing where we generate automated test cases.

The approach to Continuous Testing can vary and follow diverse pathways to ensure the best user experience is delivered, free of defects.

It’s barely viable to repeat all tests every time a new feature is added, so the Continuous Testing strategy fosters a company-wide cultural change to achieve four capabilities: Test early, test faster, test often and automate.

End-to-end test automation practices are intended to integrate QA into existing fast-paced Dev and Ops processes as a means to create continuity while maintaining faster development cycles.

It’s important to understand the role of automation in Continuous Testing. Automation alone does not enable continuity in testing, but it helps provide a qualitative assessment of risk and practice actionable tasks to mitigate these risks throughout the SDLC.

In essence, automated testing constitutes the detection process for software issues and defect prevention, whereas Continuous Testing addresses the wider challenge of improving the effectiveness of these detection sensors.

**4.1.4 CONTINUOUS DELIVERY**

It is often assumed that if we want to deploy software more frequently,we must accept lower levels of stability and reliability in our systems. In fact,peer-reviewed research shows that this is not the casehigh performance teams consistently deliver services faster and more reliably than their low performing competition. This is true even in highly regulated domains such as financial services and government. This capability provides an incredible competitive advantage for organizations that are willing to invest the effort to pursue it.

**4.1.4.1 Benefits of continuous delivery**

1. Low risk releases: The primary goal of continuous delivery is to make software deployments painless, low-risk events that can be performed at any time, on demand. By applying patterns such as blue-green deployments it is relatively straightforward to achieve

zero-downtime deployments that are undetectable to users.

2. Faster time to market: It’s not uncommon for the integration and test/fix phase of the traditional phased software delivery life cycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and

regression testing processes, developers can incorporate integration and regression testing into their daily work and completely remove these phases. We also avoid the large amounts of re-work that plague the phased approach.

3. Higher quality: When developers have automated tools that discover regressions within minutes, teams are freed to focus their effort on user research and higher level testing activities such as exploratory testing, usability testing, and performance and security testing. By building a deployment pipeline, these activities can be performed continuously throughout the delivery process, ensuring quality is built in to products and services from the beginning.

4. Lower costs: Any successful software product or service will evolve significantly over the course of its lifetime. By investing in build, test, deployment and environment automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process. products: Continuous delivery makes it economic to work in small batches. This means we can get feedback from users throughout the delivery life cycle based on working software. Techniques such as A/B testing enable us to take a hypothesis-driven approach to product development whereby we can test ideas with users before building out whole features. This means we can avoid the 2/3 of features we build that deliver zero or negative value to our businesses.

**4.1.5 CONTINUOUS DEPLOYMENT**

Continuous deployment can be thought of as an extension of continuous integration, aiming at minimizing lead time, the time elapsed between development writing one new line of code and this new code being used by live users, in production.

To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.

Instrumentation is needed to ensure that any suggestion of lowered quality results in aborting the deployment process, or rolling back the new features, and triggers human intervention.

**4.1.5.1 Expected Benefits**

The main benefits claimed for continuous deployment arise as a result of reducing lead time, with two main effects:

1. Earlier return on investment for each feature after it is developed, which reduces the need for large capital investments

2. Earlier feedback from users on each new feature as it is released to production, which affords techniques such as parallel (or A/B) testing to determine which of two possible implementation is preferred by users.

**4.1.5.2 Potential Costs**

1. Continuous deployment relies on extensive instrumentation to ensure that functionality newly made available to users does not result in incidents, lowering externally perceived quality.

2. For the same reason, continuous deployment relies on infrastructure that allows easily backing out new features when a defect has not been detected by automated tests.

**4.2** **High-Level Design**



Figure 4.2 High Level Design

**Figure** 4.2 illustrates the steps involved to achieve Site Reliability Engineering.

In a deployment environment there will be a considerable amount of traffic between individual tool servers. In order to manage these traffic, It is pretty much necessary for us to implement load balancer.

A load balancer will always take care about the loads (traffic) so we configured all our tools with the load balancer, in our project we use NGINX as our load balancer.

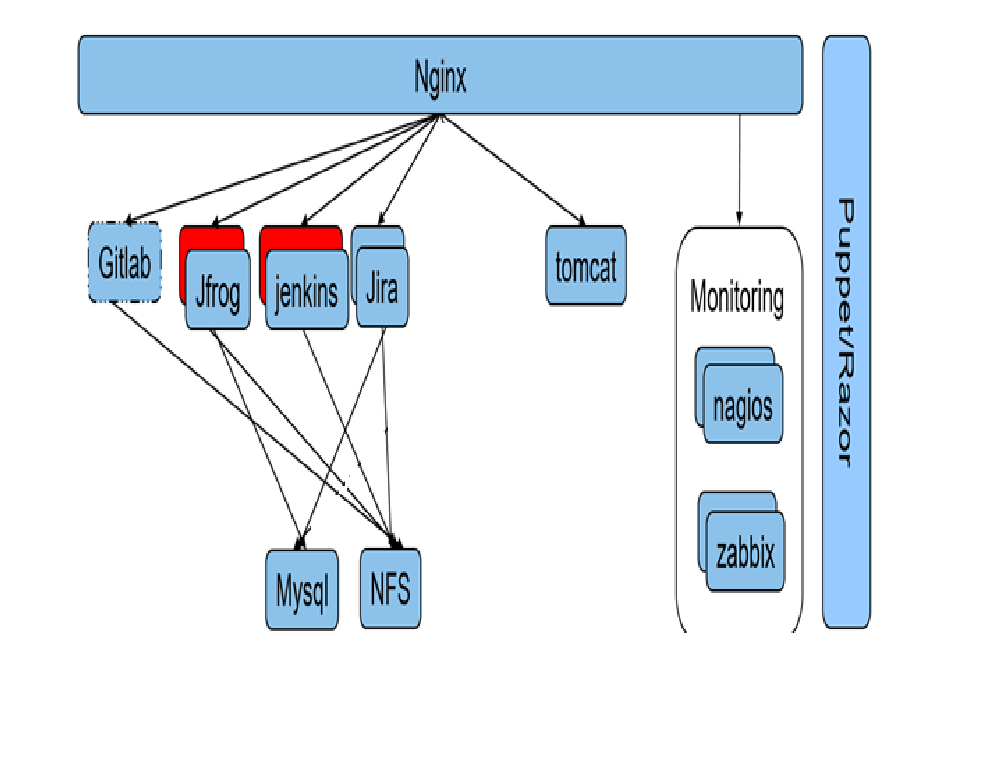


Figure 4.2.1 Load Balancer

Figure 4.2.1 Depicts a typical NGINX Load balancer which is connected on top all tool server.

NGINX (Pronounced as Engine-X) is an open source, lightweight, high-performance web server or proxy server. Nginx used as reverse proxy server for HTTP, HTTPS, SMTP, IMAP, POP3 protocols, on the other hand, it is also used for servers load balancing and HTTP Cache. Nginx accelerates content and application delivery, improves security, and facilitates availability and scalability for the busiest websites on the Internet.

Httpd service can run in two MPS (Multiple process Management)

1. Prefork( It will create a process for every request)
2. Worker (It will create a process for the first request and other request will be processed by the same PID(process id).

**4.3 Feasibility Study**

The feasibility study proposes one or more feasible conceptual solutions to the problem set of the project. The conceptual solutions give an idea of what the new system will look like. This indicates what inputs are needed by the system and what outputs will be produced.

Three things to be done to established feasibility, they are:

* First, it must be checked that the project is technically feasible.
* Second, operational feasibility must be established. It is necessary to consult the system users to see if the proposed solution satisfies user objectives and can be fitted in to current system operation.
* Third, economic feasibility must be checked. The study must determine whether the project’s goal can be achieved within the resource limits allocated to it. It must also determine whether it is worthwhile to proceed with the project at all or whether the benefits obtained from the new system are not worth the cost.
  + 1. **Technical Feasibility**

Technical Feasibility is the study of resource availability that may affect the ability to achieve an acceptable system. Technical Feasibility is the most difficult area to ensure at initial stages. Since the objectives functions and performance cannot be predicted to its fullest, everything seems possible provide, proper assumption are made. It is essential that the process of Technical Feasibility.

**4.3.2 Economic Feasibility**

Automation is highly economically feasible .The organization needed not spend much money for automation of readily available tools. The only thing is to be done is making an environment for the automation with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources .Even after the automation, the organization will not be in condition to invest more. Therefore, the system is economically feasible.

**4.3.3 Operational Feasibility**

An estimate should be made to determine how much effort and care will go into the developing of the system including the training to be given to the user. Usually, people are reluctant to changes that come in their progression. The computer initialization will certainly affected the turn over, transfer and employee job status. Hence an additional effort is to be made to train and educate the users on the new way of the system.

* + 1. **Motivational Feasibility**

An evaluation of the probability that the company is significantly motivated to support the development and implementation of the application with necessary user participation, resources, training etc. The participation and support by the organization during system study was encouraging thus eliminating any resistance in this regard. So from the Behavioural aspect the new system is supposed to have efficient from the company.

**4.3.5 Schedule Feasibility**

The time schedule required for the development of this project is very important since over-runs result is escalated projects costs and also hinders in the development of the other system.

**4.3.6 Feasibility Check**

Automation is a key to success in today’s world. Even a normal showroom vendor also wants to automate his bill statements. But it has not touched these heights till now. Since it is a new technology or not very old.

Companies are still trying their best to come up with best techniques everyday. Puppet is one of the very important invention to automate each and everything. It helps in load balancing also in high availability. As per the development life-cycle of the DevOps system is being carried out with agile methodology. This will deliver faster.

**Objectives of the Proposed System**

The objectives of this proposed system DevOps is depicted as follows,

* To automate the process of build, deploy and testing.
* To increase the rate of production releases from application and business unit stakeholders.
* To effectively automate the test and to continuously integrate the product.
* To improve the system of production and service.
* To improve quality and productivity and thus constantly decrease costs.
* To eliminate the need for massive inspection by building quality into the product in the first place.
* To automate the manual processes like dependency discovery and resolution, system construction, provisioning, update and rollback.