**E-Cart Solution**

**Project Description:** E-Cart Solution (An E-commerce web portal) is offering a (E2E) unique shopping experience to their end users. Websites is designed in robust way with High Availability (HA) mode to cope up with Risk/Disasters along with load balancer to route the traffic for better system performance. Make sure to add link to the hosted project on your preferred version control system.

**Installation / Usage Instructions:** For end to end delivery of application we need specific set of tools for implementation as mentioned below.

|  |  |  |
| --- | --- | --- |
| TOOL – NAME | PURPOSE | Dependencies |
| GIT | Version control system for Source code. | GitLab server should be up and LAN connectivity. |
| Jenkins | For CI/CD | Plugins: GIT, Ansible, Maven, Nexus, SonarSource, Selenium |
| Ansible | For Infra provisioning and deployment | Password less SSH authentication to hosts, Python, pyVmom, vmware\_guest,PiP,Vargant and Libvirt |
| Ansible Tower | For Log Monitoring/System Health check and Services monitoring | PostgreSQL, Django and Nginx and Ansible minimum 2.2 version |

**Use Cases:** Website should be dynamic in nature so that it can should have the scale in/out capability depending on the configured threshold, Load balancing among the primary and secondary servers. Instead of making whole application as one, various web- modules (i.e. user, cart checkout, payment Gateway and past orders) have been configured separately as microservices.

Below is the list of few use cases for the better infrastructure practices.

1. To support High traffic during festive season or sale time Scale IN/OUT of application shall be triggered.
2. Load balancer shall come in picture when the traffic is flooded so to avoid system slowness and crash risk.
3. Sub Modules as Microservices will be easy to debug and fast to produce results
4. Real time log analysis and system monitoring for web application.

## **The infrastructure architecture**

The overall architecture for CI/CD is depicted below.

**Plugins**

Ansible, Sonar Source, Nexus,

JENKINS FOR CI/CD

GITTTT

Developer push code Plugins



[This Photo](http://www.idatabase.in.th/2008/09/13/magic_square_c-plus-plus) by Unknown Author is licensed under [CC BY-SA-NC](https://creativecommons.org/licenses/by-nc-sa/3.0/)

Pipeline > Build>Test > Deploy QA> HTML Publisher result (Approval on the result basis ) >Prod

**Github** is where our project is hosted and where Jenkins will poll (MasterBranch) for changes to start the pipeline flow.

**SonarSource** is our source code analysis server. If anything goes wrong during the analysis (e.g. not enough unit tests), the flow is interrupted. This step is important to guarantee the source code quality index.

**Nexus**is the artifact repository. After a successful compilation, unit tests and quality analyses, the binaries are uploaded into it. Later those binaries will be downloaded by **Ansible** during the application deployment.

The **Ansible Playbook**, which is a YAML file integrated in the application source code, deploys theapplication on to a VM machine based on host entry in inventory file called hosts.

**Jenkins** is our CI/CD process orchestrator. It is responsible to put all the pieces together, resulting in the application successfully deployed in the target machine.

**State transition in Pipeline**

Exec Unit test case

Build App from Source code

Exec Integration test

Pull From GITLAB

GITLAB

Perform Quality Analysis

Artifact Upload

Approvals

Provisioning and deploy

The process starts by pulling the application source code from Github post this **Jenkins** starts building the source code using Maven.

After a successful compilation, the pipeline will perform unit tests. If nothing goes wrong during the unit tests, the pipeline flow initiates the integration tests. In this case, the test framework creates all the infrastructure needed for the tests. The integration tests are considered a success when all requests were validated against the application deployed in the test environment.

The output from unit and integration tests is a coverage report, which will be one of the artifacts used by Sonar server to generate quality metrics. The other one is the application source code. If the quality gate defined in the Sonar server is not met, the pipeline flow will fail.

If everything went fine, the flow stops its execution and waits for approval. The Jenkins server provides an interface for someone with the right permissions to manually promote the build.

**System/Log Monitoring**

Ansible tower will help us to monitor system which comes in handy with its GUI. Below is the list which can take our application performance monitoring in efficient way.

1. The Ansible Tower dashboard provides a display for system performance, Job Status going on in your Ansible environment.
2. With Ansible Tower, all automation activity is securely logged. Who ran it, how they customized it, what it did, where it happened - all securely stored and viewable later, or exported through Ansible Tower’s API.
3. We can be informed with our automation status via integrated notifications. Notify a person or team when your job succeeds, or escalate when jobs fail. Send notifications (pagerduty, SMS,Email )across your entire organization at once, or customize on a per-job basis.
4. Set up occasional tasks like nightly backups, periodic configuration remediation for compliance, or a full continuous delivery pipeline.
5. Ansible Tower's powerful provisioning callbacks allow nodes to request configuration on demand, enabling auto scaling.

**Performance and Scaling**

Performance plays a vital role to run an application in smooth way. To ensure Better performance there must be some mechanism which keeps on polling the certain metrics for which our system is targeted. In order to hit the threshold parameters infrastructure shall scale in/out(or in case of job and service restart the services/job) depending on the threshold decided.

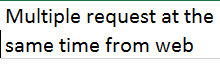
Attached is Sample snippet for AWS EC2 scale process: 

**RISK and Mitigation**

1. High Availability was missing in assignment so we need to implement this as well to make website robust in sense risk mitigation.

2. NO load balancer was introduced in problem statement so to secure the Website from unwanted traffic we need to implement a load balancer so the website should only traffic from load balancer as in round robin fashion for load sharing and security as well because firewall shall be implemented at load balancer itself.

Master Node in HA

 Load Banacer

Slave Node in HA