Leveraging the Docker Ecosystem to Provide Fast Feedback Loops for Developers

By Vivek Juneja

Fast feedback accelerates decision making, and and gives more time to making observations and identifying new directions. The OODA model (Observe, Orient, Decide and Act), pioneered by John Boyd, is increasingly used to model feedback loops in Software Development process. Successful companies like Netflix have adopted this decision making framework, and turned it into a commodity. Facebook's mantra of "Done is better than Perfect" fits this ideology of building software at a rapid pace.

Feedback loops are an essential ingredient of the Agile and DevOps mindset. It not only provides immediate result on the set of activities in a development process, but also instils positive habits among developers. Scientifically, feedback loops have also proved to change human behavior and actions.

Building a Feedback loop for Developers requires attention to 4 distinct stages. It starts with **collecting observations** including the feature usage metrics, code quality, static analysis, performance metrics and testing results, among others. Second, this information is then fed into an engine that provides valuable inferences that **summarize the observations**. These inferences avoid the clutter of raw data and produce practical metrics including test coverage, cyclomatic complexity, user preferences etc.

Third, the data generated from the first stage, and the inferences produced through the second stage are then used to **create directions**. Each such direction is a possible path of calibration and diagnosis.The last stage is where a particular **Action** is taken against the possible paths generated from the third stage.

The appropriate actions taken are then fed back into the Loop. This circular process is very effective in creating a **flow** in the software development process, and in the long run reduces the cost of changes and emergence of issues.

The most important aspect of this loop is how long it takes to complete each step since faster feedback is essential to moving quickly and producing reliable software. The goal, therefore, is to automate as much of the loop as possible.

One of the major roadblocks when building faster feedback loops is the pain of integrating and deploying large codebase. A large codebase increases the cost of compilation, bundling and deploying on an infrastructure. The cost of installing the required dependencies for a software environment to run the code also slows this process. Running on heterogeneous infrastructure in both the public cloud and local data centers leads to further latencies. All this adds up to increased decision intervals, and reaction time.

At the heart of fast feedback loops is the essence of reducing the cost of change and bad decisions. Improving the feedback time from minutes to seconds, although trivial, can have a large effect on reducing this cost of change. Developers can confidently release new infrastructure and code with the promise that no change is big enough. Putting this ideology in place requires all measures in place to minimize the visible cost of making a change.

Subhead

As communities and organizations started experimenting with microservices, they also needed a new way to speed up the feedback lifecycle from minutes to seconds. That's where Docker and its ecosystem come in. The Docker ecosystem is promises to allow developers to see their code in action by accelerating all four stages of the feedback loop into what I call a hyper-agile feedback loop.

The easiest way perhaps to understand how Docker ecosystem produces this hyper-agile feedback loop is to map to the necessary aspects of the loop:

1. Developer in a Team obtains the local test environment and the source code to their development environment. The test environment is itself a declarative manifest that is used to provision the environment for the developer. The provisioned environment is a containerized set of service instances each running independently. The local test environment contains all the necessary services and related dependencies of the Product. (*Quick and Consistent Development Environment Provisioning*)

2. Developers change or add new services, along with the required test cases. The local test cases are then run to test the changes. To avoid breaking clients, Developers could also insist on having consumer contracts to validate the change before publishing. (*Loose Coupled development*)

3. Developers then deploy these changes to the local test environment and trigger the integration tests for the various participating services that use this particular service. (*Automated Deployment and Tests*)

4. Once the changes are tested with the local test environment, the developer then commits the changes to the repository. If the service also requires a configuration change or a new dependency, the developer adds that to the environment manifest and commits the same. (*Version Controlled Development Artifacts*)

5. Committing the code leads to scheduling a build task on the build and deployment service. This service only deploys the relevant changed service as a new service version instead of replacing the old version, following the principles of canary releasing. The appropriate load balancers are informed of the new service version and a progressive load shifting takes place to allow a certain number of requests to the new changes in the service. (*Automated Build and Production Release*).

6. The new change is then observed and metrics are collected based on A/B Testing and user analytics for the change vis-a-vis the older version of the same. If the

results are promising, more load is shifted and eventually the new service takes prominence over the old one. The old version is gracefully retired. If the change is not promising, then the new service is gracefully deprovisioned, and the load is moved off from it.

The above steps merely scratch the surface of what is possible when creating a Feedback loop for Developers. The Docker ecosystem provides a rich set of services that covers a wide range of such capabilities for the developers and accelerates each of the above steps.

Using Docker platform tools such as Docker Compose and Dockerfiles, it becomes easier to package and distribute connected service oriented applications. The Dockerfiles can be version controlled, and same applies to the Docker images. A common practice for Development teams is to setup a Private Docker registry for image registration and discovery. Ecosystem products like Quay.io and Docker Hub are also worthwhile alternatives, for those looking at having SaaS solutions to docker registry.

Products like Shippable and Drone.io are rich contributors to improving the speed of Continuous Integration and thus reducing the time it takes for generating a reliable build. However, existing CI infrastructure like Jenkins can integrate with the Docker image creation and publishing process to deliver Build and Deploy pipeline.

The most interesting aspect of the Docker ecosystem is the rise of niche PaaS options like Tsuru, Giantswarm and Deis among others. Some of these PaaS options are transparent solutions, while others provide a layer of abstraction in the form of Developer friendly DSLs to architect and deploy complex applications. With rapid change cycles and iterative development process, a PaaS solution like these come handy when considering quick developer feedback. Existing PaaS implementations like CloudFoundry and OpenShift have already geared up to integrate with the Docker toolchain.

Most of the organizations adopting Docker containers for speeding up their development process have the inherent need of managing multiple versions of their services at the same time. This is primarily needed to support Canary releasing and Service Immutability. The latter is an interesting proposition, where it is considered to have a change rather deploy a new service instead of updating an existing one. This reduces the time taken to rollback and provides opportunity A/B testing. One such solution in the Docker ecosystem is VAMP. The product allows declarative expression of Service deployment and Routing, and sits between the application and the underlying PaaS. This allows for Netflix style Canary Releasing, A/B Testing of new services and Service SLA Management.

If Programmable infrastructure is still far off within your enterprise, it is a good time too look out for products like Tutum, Google Container Engine and Amazon ECS that allows hosting containerized applications.

A key part to create feedback loops is Monitoring. Tools such as Scout and Data Dog are mature options in this space. CAdvisor and Docker stats are also convenient Open alternatives for monitoring containers. Together with traditional Application monitoring tools like New Relic, these options are valuable to create a source of metrics for Developers.

Validating each Docker container to ensure that it sticks to the agreed upon company policies and governance rules is also critical for organizations of all types. This allows development teams to be able to make changes in container configuration with certainty. This is where Product like GuardRail is essential to be able to provide constant governance capabilities to a Developer pipeline.

At the heart of all these concerns for Fast Feedback loop is the idea of Standardization. To enable fast feedback for changes made by Development team, it is essential to provide them with template that has all the underpinnings of the various Docker ecosystem toolchain, including Monitoring, Governance, PaaS integration and instrumentation.

Projects like ZeroToDocker are initial steps to create a boilerplate for Developers that is already preconfigured to provide Micro Services based development practices. This allows Developers to invest their time developing and releasing reliable code instead of managing the nitty-gritties of a Service oriented system.

The idea is to have a standardized "hello-world" sort of boilerplate project for different language types that consolidate the best practices of a Service oriented development including preconfigured logging, monitoring, service registration and discovery, and integration with Docker ecosystem toolsets. This forms the Service Unit of a Micro Services system. A Service Unit of this type can be quickly developed and released without re-inventing the wheel for each Developer.

The Docker Container format and the ecosystem is still in its early days, and there is more to come in order to create a mature platform for fast feedback loop. However, it does provide building blocks for creating the Hyper-Agile feedback loop and indicates the taste of how software will get built and released in the coming years.

Vivek Juneja, an engineer based in Seoul, is focused on cloud infrastructure and microservices. He started working with cloud platforms in 2008, and was an early adopter of AWS and Eucalyptus Cloud. He's also a technology evangelist and speaks at various technology conferences in India. He writes @ www.cloudgeek.in and www.vivekjuneja.in, and loves grooming technology communities. You can also reach him by email: vivekjuneja@gmail.com.