INTRODUCTION

The MeerKAT is a 64-dish radio telescope situated in the remote Karoo desert region of South Africa and is the reference implementation for the Square Kilometre Array, which will be the largest telescope in the world. It has a sophisticated control and monitoring system that has leveraged virtualisation technologies and automation from the start. This has enabled the adoption of software development practices such as **continuous integration** and **automated acceptance testing** which allowed for rapid, incremental development of the system. Despite this, some last mile problems were encountered around release which prompted efforts to further optimise its build and deployment process.

THE "LAST MILE"

Issues that crop up between integrated changes (code that has been merged into trunk/mainline) and their release to the production environment. These problems result in failures, and troubleshooting efforts during release time. Some informal analysis revealed the following issues:

1. Repeatable deployments not guaranteed due to
   a. unpinned transient dependencies
   b. late convergence of system baseline state by imperative deployment scripts
2. Automated testing did not exercise full deployment procedure
   a. tests ran in a static, long-lived environment/node
3. Unclear separation of **build** and **run** stages
   a. version control release branching scheme used to manage deployments

**Continuous Integration**

A change to each subcomponent triggers an automated build that is tested by a suite of unit tests. Any changes that pass the tests pass the **travis-ci.org** and are merged to the master branch. A **Jenkins CI server** will then provision the pipeline to our private **PaaS** environment. The latest build version of the six leaf packages in the CAFs dependency tree is to be used for dependency resolution in the next stage.

**Pinned Dependencies**

Dependence is resolved and pinned to specific versions by **pip-compile**. This is a complete environment file with all Python dependences that satisfy the requirements of the most recent environment builds of system, proxy, kernel, astropy, lcs, astrom, and others. This will explicitly depend on the next leaf of the CAF Python packages that could potentially make the coherent system.

**Candidate Image**

A **LCF template** that incorporates a potential-release of the CAF system. This includes all dependencies, and configuration derived from a specific build pipeline run. Every candidate can be described in a LCF template, which can then be validated by our Automated **Qualification Framework** (integration tests and acceptance tests).

**Node Configuration**

The node configuration describes a system in terms of a number of nodes that belong to it and the roles that each of these nodes play. The deployment scripts will run these operations based on the roles in order to specialise them.

**Homogenous Nodes**

Nodes are created from the same image and are only differentiated during deployments based on the **master_seed**. This is a key design principle that greatly simplifies the release process.

**Repeatable Builds And Deployments**

The same candidate and configuration configuration will produce the same system. Furthermore, a set of pinned requirement produces the same candidate.

This property of determinism allows us to increase confidence in changes.