# Understanding Apache Brooklyn

# CAMP

**Abstract:**

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

*Key Points: Portability between clouds” is emerging as one of the primary concerns of cloud computing*. By standardizing the management API for the use cases around deploying, stopping, starting, and updating applications, this specification increases consumers’ ability to port their applications between PaaS offerings.

Management API of PaaS:

* Building
* Running
* Administration
* Monitoring
* Patching of Applications

Platform as a Service defines the self-service management API to the consumer. The API is an interface into a platform implementation layer that controls the deployment of applications and their use of the platform.

Self-Service Platform API (Platform Implementation):

* Runtime Containers
* Firewalls
* Load Balancers
* Web Servers
* Databases
* Message Queues
* Log Aggregators

The platform implementation is a management client of the underlying resources that transforms (through policies) the application requirements expressed by the Application Administrator into provisioning and other operations on those resources.

The Platform Administrator manages the underlying hardware, storage, networks, and software services that make up the platform through existing administrative interfaces.

Management APIs provides the essential elements that give control over the deployment, execution, administration and metering of their application and its deployment environment.

**Resources:**

* *Platform*

The platform resource is the primary view of the platform and what is running on it. References collections of resources that represent services provided by the platform, the applications running on this platform, as well as collection of metadata resources that describe the resources supported by the platform as well as extensions.

* *Assemblies*

Assembly resource represents running applications. Operations on assembly resource affect the components and elements of that application.

* *Components*

An assembly resource is comprised of one or more component resources. A component resource represent can a dynamic element of an application such as deployed Ruby Gem.

* *Plan*

Plan is the meta-data that provides a description of the artifacts that make up an application, the services that are required to execute or utilize those artifacts, and the relationship of the artifacts of those services. Plan can be represented in YAML format.

* *Services*

A service resource represents a blueprint for creating component resources that utilize or embody a platform-provided service. For ex a Service may represent the platform’s ability to create a message queue for use by an application.

**Operations and Sensors:**

Operations and sensors provide a way of interacting with an application through the CAMP API. An *operation resource* represents actions that can be taken on a resource; the sensor resources represent dynamic data about resources, such as metrics or state. A sensor resource is useful for exposing data that changes rapidly, or that might need to be fetched from a secondary system. A sensor resource can also offer Operations to allow resetting metrics, or adjusting frequency collection settings. Multiple operation resources and sensor resources can be exposed both on assembly resources and component resources. Operations are also known as effectors. The combination of Operations and Sensors enables ongoing management. This can include automation techniques such as using policies, event-condition-action paradigms, or autonomic control. A Consumer can use the REST API to perform such management. A Provider can also use them. For example, a component resource could be offered that allows for “autoscaling” capacity based on the volume of work an application performs.

**Deployment**

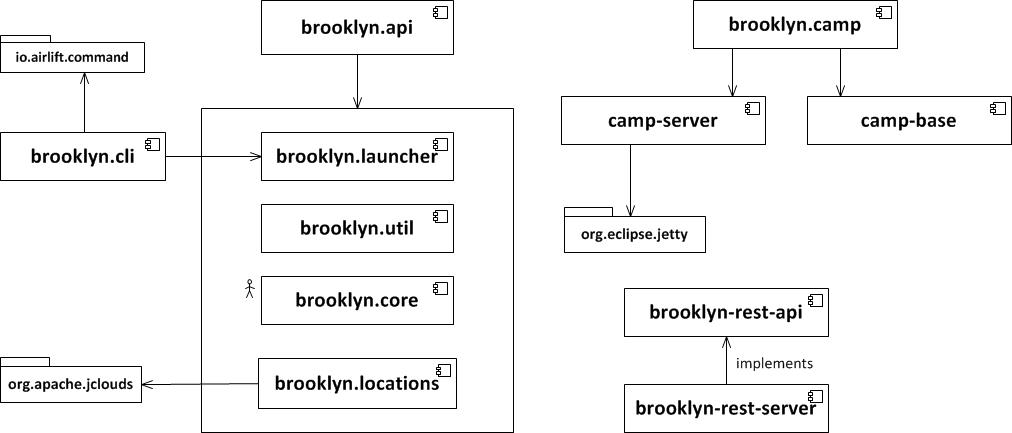
Platform Deployment Package (PDP) is an archive containing a Plan file together with application content files such as web archives, database schemas, scripts, source code, localization bundles, and icons; and metadata files such as manifests, checksums, signatures and certificates. It can be used to move an Application and its components from Platform to Platform or between an Application Development Environment and a Platform.

# Apache Brooklyn Introduction

## Few Concepts

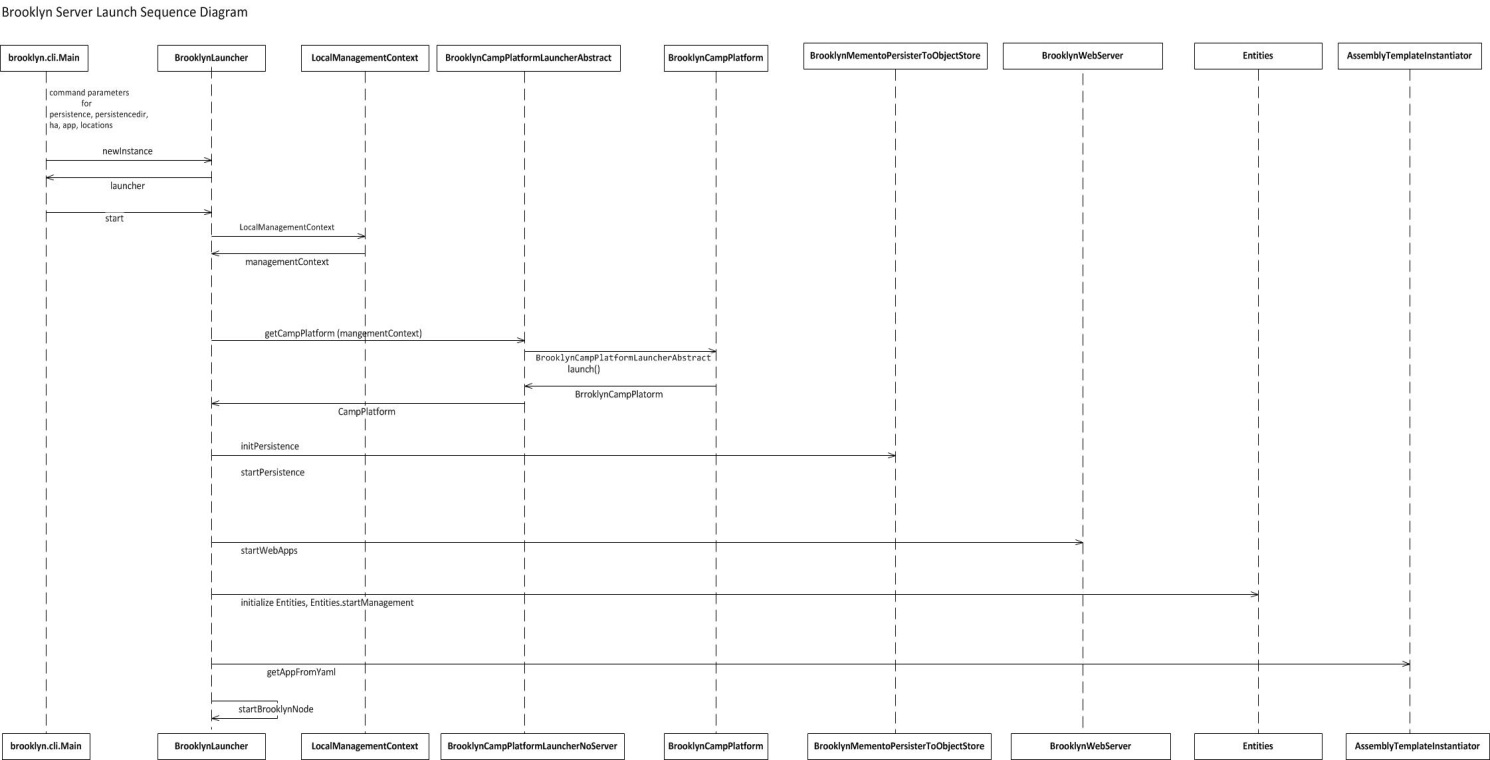
**Autonomic Computing and Promise Theory**

## High Level Project Structure

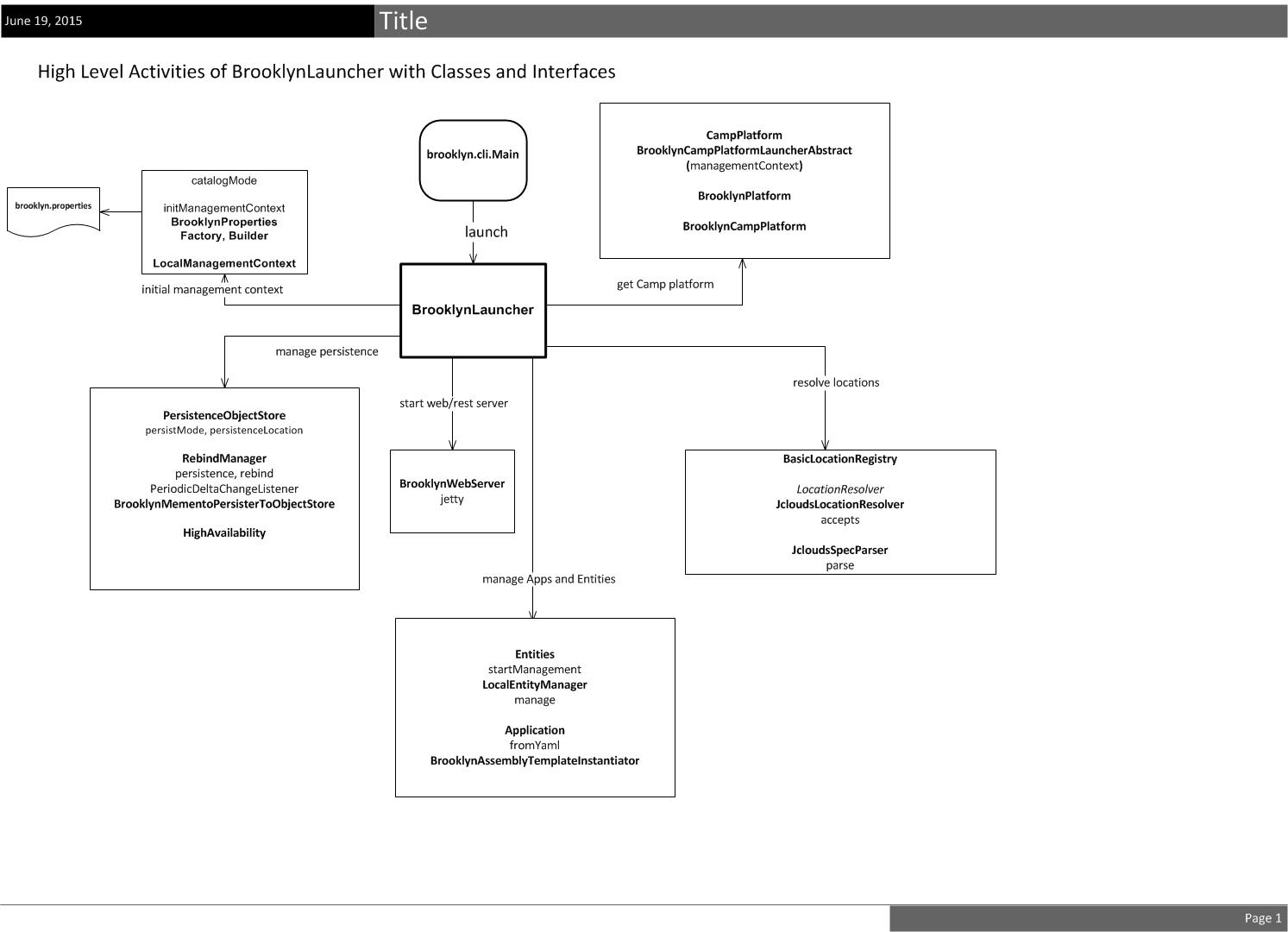


# How Brooklyn is launched:

* BrooklynLauncher
* ManagementContext
* RebindManager
* Persistence
* Location Resolutions
* Catalogs and Applications

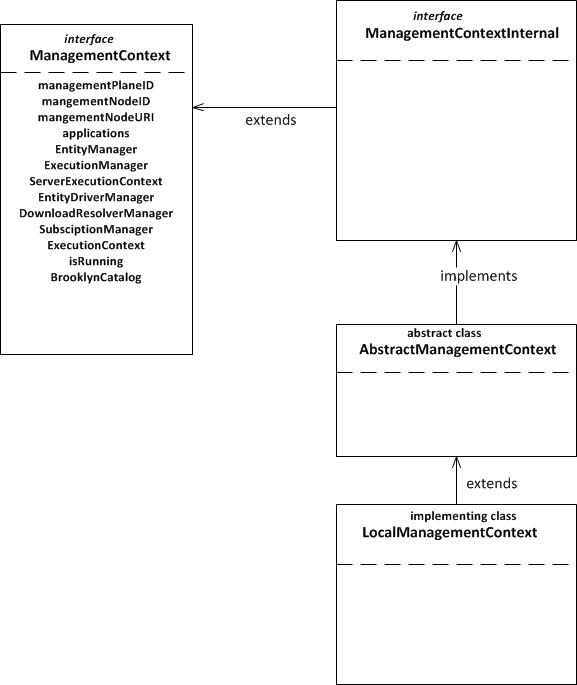


### Brooklyn Launcher Key Activities and related Classes



### ManagementContext

It’s the kind of heart of the application. It has all the information of the Brooklyn server including the Applications and Entities.



When new entities are created, the entity is wired up to an application by giving it a parent. The entity is then explicitly “managed”, which allows other entities to discover it.

Typically a Brooklyn deployment has a single management context which records:

* all entities under management that are reachable by the application(s) via the parent-child relationships,
* the state associated with each entity,
* subscribers (listeners) to sensor events arising from the entities,
* active tasks (jobs) associated with any the entity,
* which Brooklyn management node is mastering (managing) each entity.

## File-based Persistence

To persist to the file system, start brooklyn with:

brooklyn launch --persist auto --persistenceDir /path/to/myPersistenceDir

## Object Store Persistence

Brooklyn can persist its state to any Object Store API that jclouds supports including S3, Swift and Azure. This gives access to any compatible Object Store product or cloud provider including AWS-S3, SoftLayer, Rackspace, HP and Microsoft Azure.

To configure the Object Store, add the credentials to ~/.brooklyn/brooklyn.properties such as:

brooklyn.location.named.aws-s3-eu-west-1:aws-s3:eu-west-1

brooklyn.location.named.aws-s3-eu-west-1.identity=ABCDEFGHIJKLMNOPQRSTU

brooklyn.location.named.aws-s3-eu-west-1.credential=abcdefghijklmnopqrstuvwxyz1234567890ab/c

**References:**

[**https://brooklyn.incubator.apache.org/**](https://brooklyn.incubator.apache.org/)

[**https://github.com/airlift/airline**](https://github.com/airlift/airline)

[**https://jclouds.apache.org/**](https://jclouds.apache.org/)

[**http://docs.oasis-open.org/camp/camp-spec/v1.1/cs01/camp-spec-v1.1-cs01.pdf**](http://docs.oasis-open.org/camp/camp-spec/v1.1/cs01/camp-spec-v1.1-cs01.pdf)

[**http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf**](http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf)