

apache **brooklyn**

What it is and why you might use it

Richard Downer
richard@apache.org
Presented at ApacheCon Europe 2014

Hello to those watching from home. The speaker's notes on most slides will provide more information about what is being discussed. Some of the content is in the form of demonstrations; the slides contain links to the recordings which you can watch online.

A brief history of Brooklyn

- Brooklyn was started by Cloudsoft (my employer)
- Open sourced in April 2012 (Apache 2 license)
- Joined the Apache Incubator in May 2014

The only slide about the company

- Cloudsoft use Apache Brooklyn was the base for AMP: Application Management Platform
- Provide commercial support and special integrations for Brooklyn
- Brooklyn is not “open core”: Cloudsoft is committed to a comprehensive and expanding Apache Brooklyn feature list and codebase

...but we won't talk about Cloudsoft any more

The problem with Brooklyn:

**How to
describe it
in one
sentence**



The main thing I struggle with in Brooklyn - how to describe it in one sentence - the so-called “elevator pitch”. It’s not easy to come up with a summary which describes Brooklyn accurately, concisely, and in a way which captures Brooklyn’s unique advantages.

So how do others describe Brooklyn? Let’s take a look...

apache **brooklyn**



**Application modelling,
monitoring and management**

Here we see some attempts to describe Brooklyn succinctly.

apache **brooklyn**



BLUEPRINT FOR SUCCESS!

**Model, monitor and manage your applications
with policy-based automation**

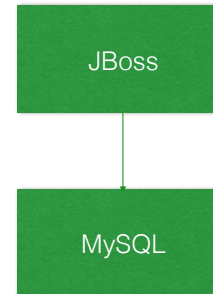
Blueprints are apparently something important!

These taglines

- Are accurate
- Are short (mostly)
- Fail to provide any useful insight into what Brooklyn actually does

Let's describe by example

- A simple web application:
 - JBoss web app container
 - MySQL database



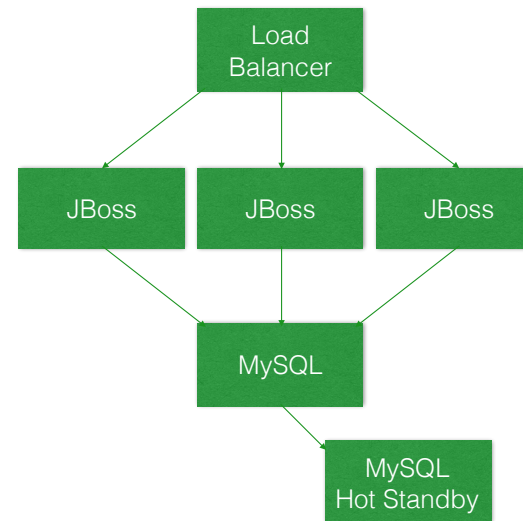
Let's say that we are the developers for a moderately complex web application. It needs a Java web application container for the front end, and a MySQL database for the backend.

In development...

- You could set up your JBoss and MySQL:
 - Run them on localhost
 - Use Vagrant to start to two virtual machines
 - Provision a couple of cloud instances
- JBoss needs to locate MySQL
 - Easy enough to configure this by hand

But what about QA and production?

- A single web server and a single database is not a production grade configuration!
- It's not scalable: it can't handle heavy load
- It's not resilient: it can't handle failures
- So we add multiple JBosses
- ...and a load balancer
- ...and a MySQL standby node



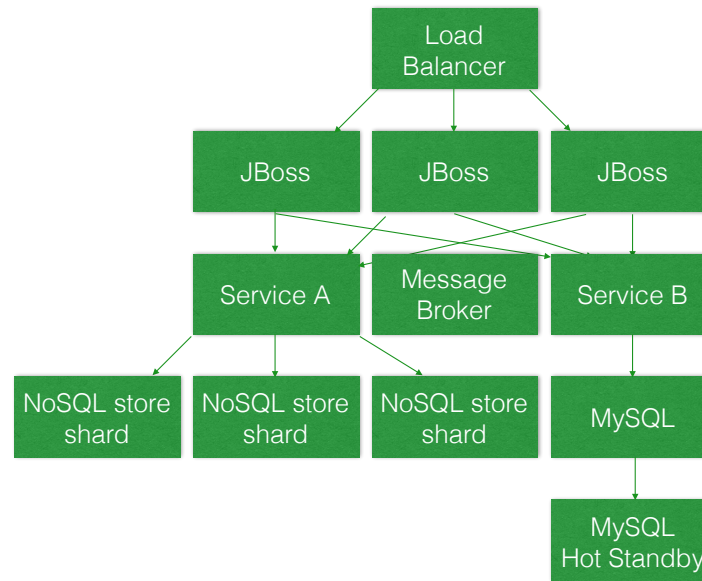
So this is a lot of things to be wired up!

1. Everything needs a server to run on, and something needs to set up that server and install the software
2. MySQL needs to know about the hot standby
3. Each JBoss needs to know about the location of and credentials for MySQL
4. The load balancer needs to know about the location of every JBoss

But wait, there's still more

- This is just for a simple application of a web server and a database!
- What about something with:
 - Multiple tiers and services
 - Connected by a message queue
 - With multiple types of database backend - SQL and NoSQL

More complexity



The first problem

- How do you deploy an app with multiple components?
- How do you get the servers to deploy onto?
- How do you get the software onto servers?
- How do you configure the software pieces to talk to each other?
- How do you make this process fast, easy and automatable?

With apache **brooklyn**, of course!

The First Pillar of Brooklyn

Deployment and wiring



Obligatory page full of logos



So far we've used JBoss and MySQL as an example - but there are many more applications that Brooklyn supports out-of-the-box. And many of these can have complex deployment topologies, which Brooklyn is able to support - such as sharded databases, or databases whose resilience and sharding depend on a knowledge of the data centre arrangements.

Blueprints: how to deploy with Brooklyn

- Describe your application to Brooklyn -
make a blueprint
 - Describe the components you are using
 - Describe how they must be configured
 - Describe how they relate to each other
 - Describe where they are to be deployed

Blueprint for our simple example

```
name: My Web Application
location: AWS_eu-west-1
services:

- serviceType: brooklyn.entity.database.mysql.MySqlNode
  id: db
  name: My DB
  brooklyn.config:
    creationScriptUrl: https://bit.ly/brooklyn-visitors-creation-script

- serviceType: brooklyn.entity.webapp.jboss.JBoss7Server
  name: My Web
  brooklyn.config:
    wars.root: http://bit.ly/brooklyn-example-helloworld-war
  java.sysprops:
    brooklyn.example.db.url: >
      $brooklyn:formatString("jdbc:%s?s?user=%s\\&password=%s",
        component("db").attributeWhenReady("datastore.url"),
        "visitors", "brooklyn", "br00k11n")
```

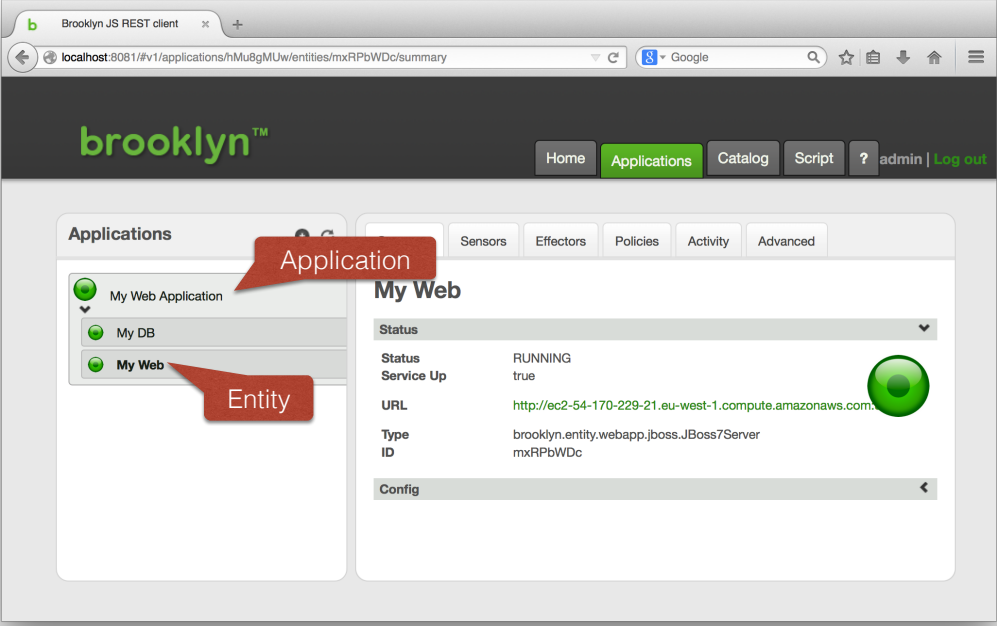
1. This is what a blueprint looks like - it's a document written in YAML.
- 2.The blueprint has a name...
- 3....and a location. This example is deploying to a location which represents Amazon AWS EC2, region eu-west-1
- 4.Next is a list of services. The first service is the database.
- 5.The service type is a Java class (often, but not always) - this class knows all about the software!
- 6.It has an ID (optional) and a name.
- 7.It also has configuration. This particular configuration references a SQL “creation” script. The MySQL entity class will read this resource and run it as SQL on the database.
- 8.Next we define the web server. It has similar arrangements
- 9.The example db url is special - it calls a function, which references the database by ID, and does something called “attribute when ready”?
- 10.To understand “attribute when ready”, it's best to see this in action.

apache **brooklyn**

A demonstration of deployment

Watching at home? Go to:
<http://youtu.be/0iJ18tla0Qk>

Anatomy of an application



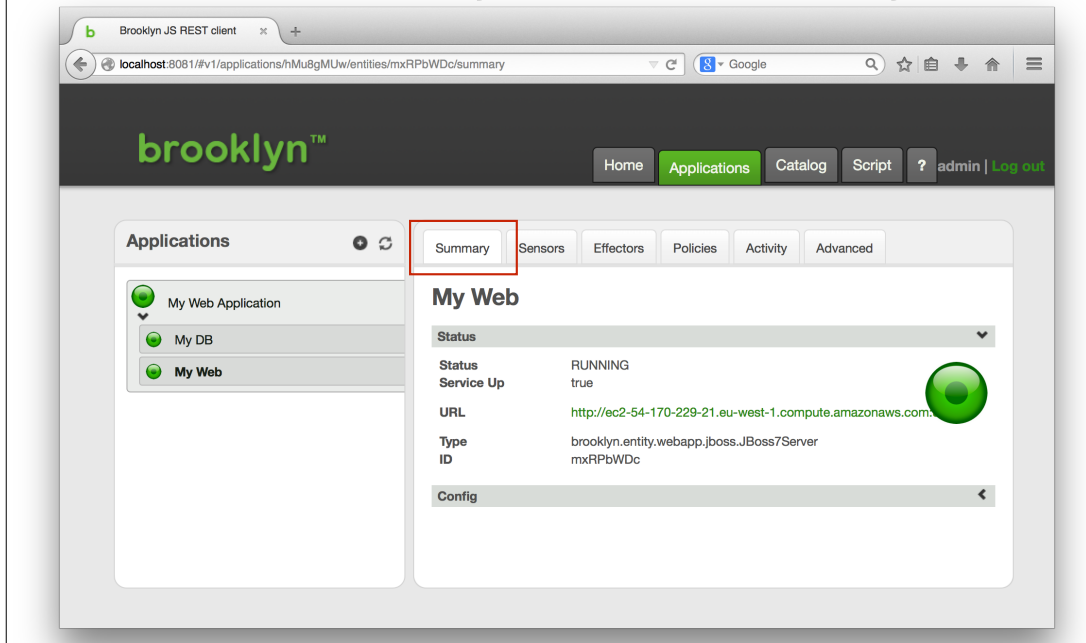
The screenshot shows the Apache Brooklyn web console interface. The browser address bar indicates the URL: `localhost:8081/#v1/applications/hMu8gMUw/entities/mxRPbWdc/summary`. The console header includes the Brooklyn logo and navigation links: Home, Applications (highlighted), Catalog, Script, and admin | Log out. The main content area is titled 'Applications' and features a tree view on the left with three items: 'My Web Application', 'My DB', and 'My Web'. A red callout box labeled 'Application' points to the 'My Web Application' item. The 'My Web' item is expanded, showing a detailed view with a red callout box labeled 'Entity' pointing to it. The detailed view includes a 'Status' section with a dropdown menu, a 'Service Up' indicator (true), a 'URL' field with a green status icon, and a 'Type ID' field. Below this is a 'Config' section with a dropdown arrow.

| My Web | |
|------------|---|
| Status | RUNNING |
| Service Up | true |
| URL | http://ec2-54-170-229-21.eu-west-1.compute.amazonaws.com |
| Type ID | brooklyn.entity.webapp.jboss.JBoss7Server mxRPbWdc |
| Config | |

Here we'll describe some of the parts of the Brooklyn web console. In technical terms, Brooklyn is controlled by a REST API, and this is a Javascript GUI that interacts with the REST API. So anything that can be done in the UI, can also be done by another tool interacting directly with the API, making Brooklyn a very powerful, scriptable component part of a larger system, if required.

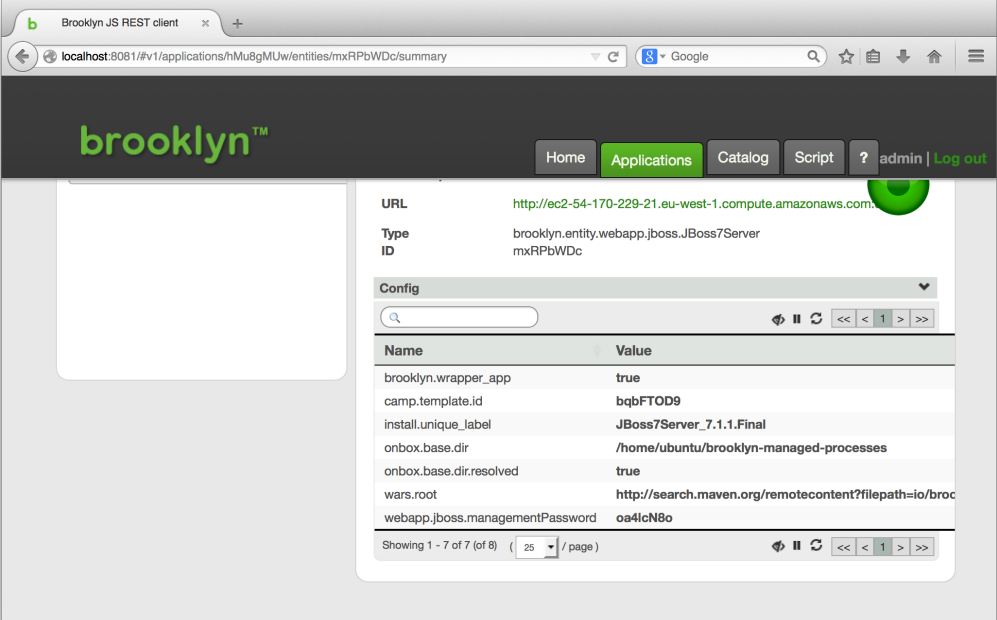
We can see here the “tree” structure that has an application as its root and entities as branches and leaves - although this particular example only has a single level of entities. (We'll see a deeper example later on.)

Anatomy of an entity



The summary tab shows key information about the entity or application. For a web server, the URL is “promoted” to the summary page so it is easily accessible.

Anatomy of an entity



The screenshot shows a web browser window with the Brooklyn REST client interface. The browser address bar shows the URL `localhost:8081/#v1/applications/hMu8gMUw/entities/mxRPbWdc/summary`. The interface features a dark header with the Brooklyn logo and navigation links: Home, Applications (highlighted), Catalog, Script, and admin | Log out.

The main content area displays the following information:

- URL:** `http://ec2-54-170-229-21.eu-west-1.compute.amazonaws.com`
- Type:** `brooklyn_entity.webapp.jboss.JBoss7Server`
- ID:** `mxRPbWdc`

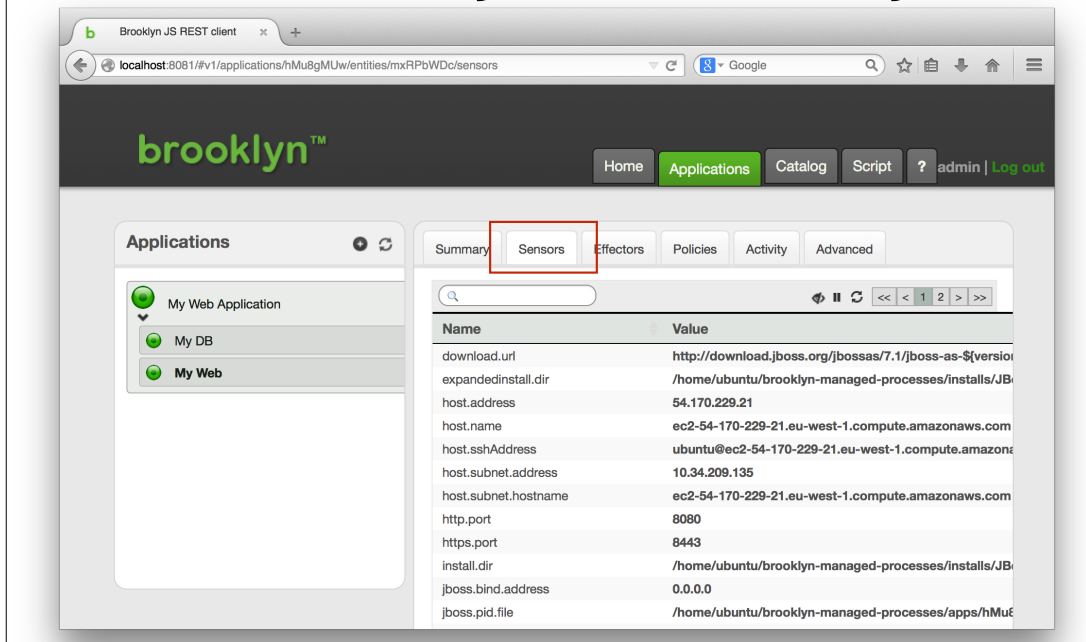
The "Config" pane is expanded, showing a table of configuration properties:

| Name | Value |
|--|---|
| <code>brooklyn.wrapper_app</code> | <code>true</code> |
| <code>camp.template.id</code> | <code>bqbFTOD9</code> |
| <code>install.unique_label</code> | <code>JBoss7Server_7.1.1.Final</code> |
| <code>onbox.base.dir</code> | <code>/home/ubuntu/brooklyn-managed-processes</code> |
| <code>onbox.base.dir.resolved</code> | <code>true</code> |
| <code>wars.root</code> | <code>http://search.maven.org/remotecontent?filepath=io/broc</code> |
| <code>webapp.jboss.managementPassword</code> | <code>oa4lcN8o</code> |

At the bottom of the config pane, it indicates "Showing 1 - 7 of 7 (of 8) (25 / page)".

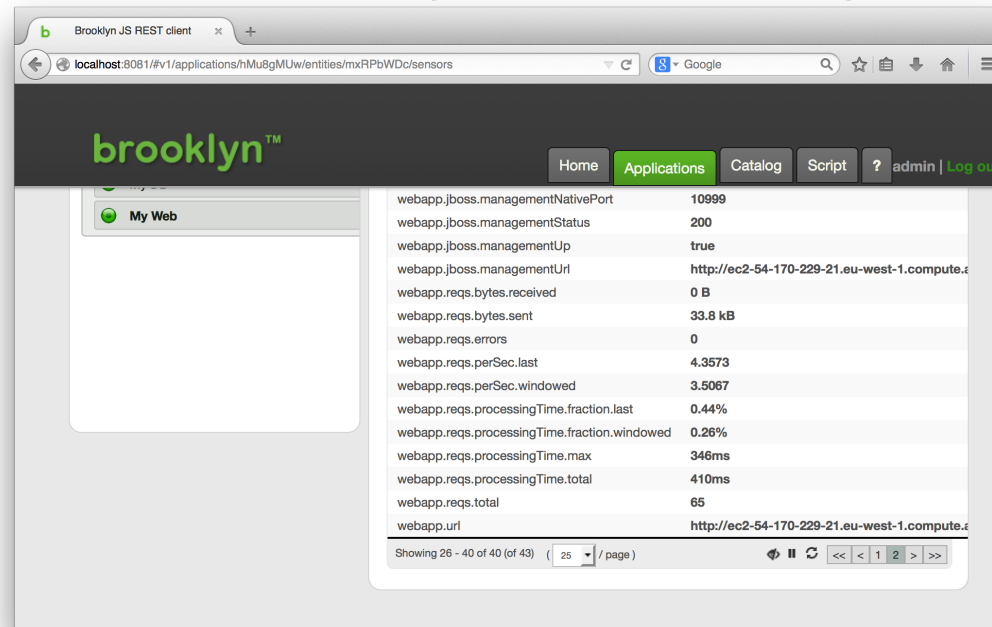
By expanding the “Config” pane, we can see some of the configuration that the entity has.

Anatomy of an entity



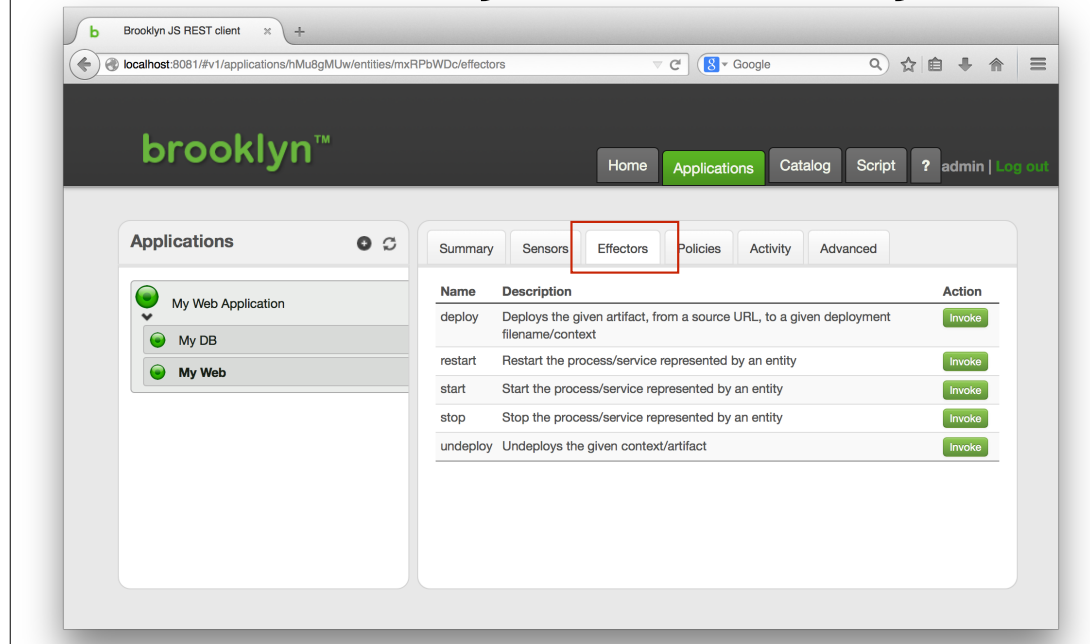
The “Sensors” tab shows detailed information about the entity. A sensor is something that the entity measures and then publishes. Here we can see some static information about the server, but if we scroll down...

Anatomy of an entity



...we can also see some dynamic sensors returning metrics about the entity, such as the number of requests and the amount of data moved. These are updated frequently!

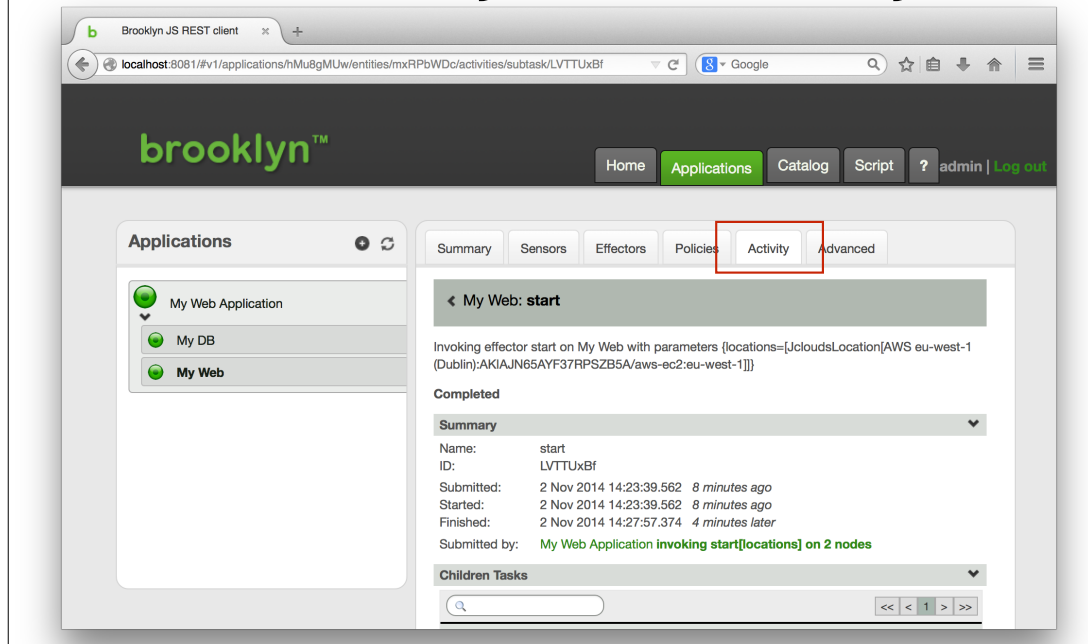
Anatomy of an entity



Effectors are something that causes the entity to change in some way. “Stop” and “start” are the most obvious effectors, and these are supported by almost all entities. The web server entities also have a “deploy” effector, which allows another web application to be loaded onto the server.

We’ll skip the policies tab for now - more on that later.

Anatomy of an entity



Activity, as you saw in the demo video, shows what tasks each entity is performing. Effector calls result in a task; and tasks can be split into sub-tasks. The activity view allows you to explore this hierarchy and find out the details about the success or failure of each task.

apache **brooklyn**

Make it a load-balanced web cluster

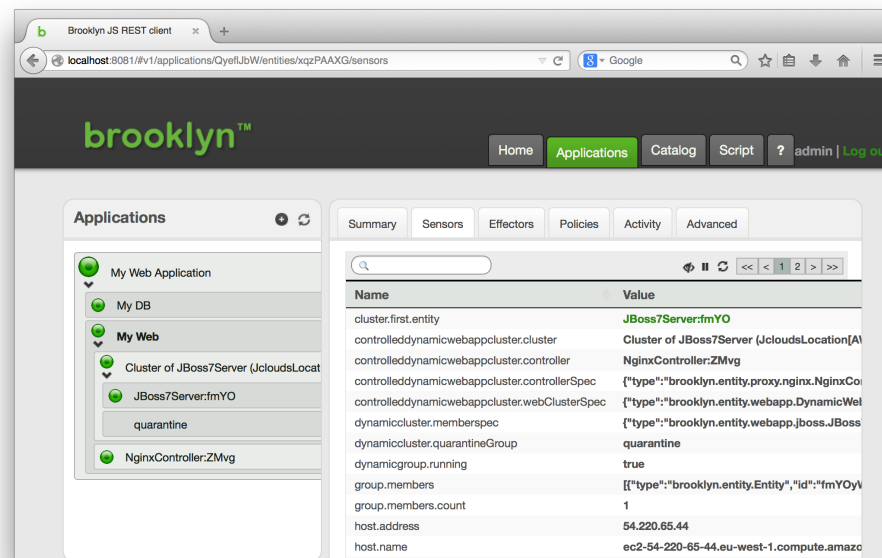
```
name: My Web Application
location: AWS_eu-west-1
services:

- serviceType: brooklyn.entity.database.mysql.MySqlNode
  id: db
  name: My DB
  brooklyn.config:
    creationScriptUrl: https://bit.ly/brooklyn-visitors-creation-script

- serviceType: brooklyn.entity.webapp.ControlledDynamicWebAppCluster
  name: My Web
  brooklyn.config:
    wars.root: http://bit.ly/brooklyn-example-helloworld-war
  java.sysprops:
    brooklyn.example.db.url: >
      $brooklyn:formatString("jdbc:%s%s?user=%s\\&password=%s",
        component("db").attributeWhenReady("datastore.url"),
        "visitors", "brooklyn", "br00k11n")
```

So we've demonstrated a single-web-server topology. What if we wanted to make this a cluster of web servers with a load balancer? It's incredibly easy - just a single change is required. The JBoss7Server class, and the ControlledDynamicWebAppCluster class both support the same interface - so by changing this one thing, we get the advanced behaviour we need.

Make it a load-balanced web cluster



The screenshot shows the Apache Brooklyn web console interface. The browser address bar indicates the URL is `localhost:8081/#v1/applications/QyefJbW/entities/xqzPAAAXG/sensors`. The page title is "brooklyn™". The navigation menu includes "Home", "Applications", "Catalog", "Script", and "admin | Log out".

The main content area is titled "Applications" and shows a tree view of the application structure:

- My Web Application
 - My DB
 - My Web
 - Cluster of JBoss7Server (JcloudsLocat
 - JBoss7Server:fmYO
 - quarantine
 - NgixController:ZMvg

The right-hand pane shows the "Summary" tab for the selected entity. It displays a table of properties and their values:

| Name | Value |
|--|--|
| cluster.first.entity | JBoss7Server:fmYO |
| controleddynamicwebappcluster.cluster | Cluster of JBoss7Server (JcloudsLocation[A |
| controleddynamicwebappcluster.controller | NgixController:ZMvg |
| controleddynamicwebappcluster.controllerSpec | {"type":"brooklyn.entity.proxy.nginx.NgixCo |
| controleddynamicwebappcluster.webClusterSpec | {"type":"brooklyn.entity.webapp.DynamicWel |
| dynamiccluster.memberspec | {"type":"brooklyn.entity.webapp.jboss.JBoss |
| dynamiccluster.quarantineGroup | quarantine |
| dynamicgroup.running | true |
| group.members | [{"type":"brooklyn.entity.Entity","id":"fmYOyV |
| group.members.count | 1 |
| host.address | 54.220.65.44 |
| host.name | ec2-54-220-65-44.eu-west-1.compute.amazo |

So in this view we can see that the JBoss entity has been replaced by several more. There is a Cluster of JBoss7Server, which is where our cluster now lives. Because this is a “dynamic” cluster, it’s size can be changed, and there are effectors available which can do this. A “controller” cluster means that there is a controller which provides a front-end to the cluster; in this case it’s NGiNX which is acting as a load balancer. NGiNX is automatically configured with the details of the servers inside the cluster. And if the contents of the cluster change, NGiNX is reconfigured.

Locations

- Fully “cloud aware” using Apache jclouds



- Amazon EC2, CloudStack, OpenStack, SoftLayer, Google GCE. ...
- Provisions instances on demand, installs and customises; de-provisions when no longer required
- Or use “BYON” - bring your own nodes
- Or, for testing, deploy to localhost

Locations are one of the most important concepts in Brooklyn. Right from the beginning it was designed to be cloud-aware, and cloud-agnostic, and as a result it's naturally suited to deploying to many different cloud providers. Our way of doing this is with Apache jclouds, which is a Java multi-cloud toolkit - it is Brooklyn's single most important dependency!

With jclouds, your application can deploy to Amazon AWS, as demonstrated, and also CloudStack, OpenStack, SoftLayer, Google Compute Engine, and more.

Cloud instances are provisioned when needed, software installed and then customised. When the entity is no longer needed, it is automatically deprovisioned.

If you have your own “metal” which you prefer to use, then Brooklyn can be configured with a BYON provider - “bring your own nodes”. Or, for testing your application, you can simply choose to use localhost.

Locations

- **Multiple locations**
 - **Multi-geography deployments reduce latency to International users**
 - **“Fabrics” replicate an application topology into different regions**
 - **Geography-aware DNS routes visitors to closest server**
- **Availability zone awareness**
 - **Clusters can distribute their members across availability zones**

And Brooklyn is not limited to deploying a blueprint in a single location - multiple locations can be used! A fabric will replicate a topology in different locations; this could then be tied to a geography-aware DNS which routes users to the location nearest to them.

It also allows for resilience, allowing you to have working services in different data centres, even with different cloud providers if you choose. Some entities have a deeper level of knowledge, such as clusters which are availability-zone-aware and distribute their members in different zones, and support for Cassandra “snitches” which introduce a degree of availability-zone-awareness to Cassandra for maximum effectiveness.

The Second Pillar of Brooklyn

Deployment and wiring Runtime management



What is runtime management?

- Deployment is merely the opening move in the game
- Runtime management is...
 - Instructions to change the deployed application
 - Monitor the health of all the components and react to failures
 - Monitor the load on the components and react to rising and falling demand
 - Optimise for cost, responsiveness, and more

Policies

- **Something that will make changes to the application without requiring operator intervention**
- **Policies are attached to an entity**
- **Monitor the entity's sensor data and other information**
- **Makes changes to the application by invoking effectors on the entity**

apache **brooklyn**

Demonstration and descriptions of some policies

Watching at home? Go to:
<http://youtu.be/-WdbiDpZ8-g>

Blueprint for a policy

```
brooklyn.policies:  
- type: brooklyn.policy.ha.ServiceReplacer  
- type: brooklyn.policy.autoscaling.AutoScalerPolicy  
brooklyn.config:  
  metric: $brooklyn:sensor("brooklyn.entity.webapp.ControlledDynamicWebAppCluster",  
"webapp.reqs.perSec.windowed")  
  metricLowerBound: 10  
  metricLowerBound: 100  
  minPoolSize: 2  
  maxPoolSize: 5  
  dynamiccluster.zone.enable: true  
  dynamiccluster.numAvailabilityZones: 2  
memberSpec:  
  $brooklyn:entitySpec:  
    type: brooklyn.entity.webapp.jboss.JBoss7Server  
    brooklyn.enrichers:  
    - type: brooklyn.policy.ha.ServiceFailureDetector  
    brooklyn.policies:  
    - type: brooklyn.policy.ha.ServiceRestarter  
      brooklyn.config:  
        failOnRecurringFailuresInThisDuration: 30 minutes
```

This fragment of YAML should be appended directly onto the end of the blueprint we used earlier (but note that it has a two-space indent as it is part of the My Web entity!)

Here we add two policies to the cluster - a service replacer, and an autoscaler. The autoscaler is connected to a sensor, and the configuration defines the thresholds and limits that the policy will operate to. We also enable options to make the cluster availability zone aware.

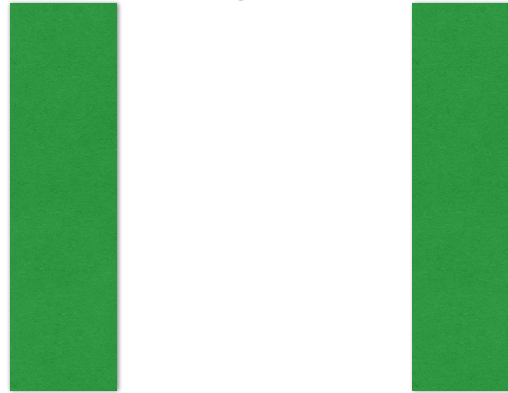
Member spec defines details about the entities that are used to fill the cluster - JBoss7 servers in this case. We are adding to each JBoss 7 entity an enricher called ServiceFailureDetector. An enricher is something that processes sensor data and publishes another sensors; in this case, it is monitoring key JBoss7 entity sensors and publishing a new sensor which simply says if the entity is healthy or failed. Finally, we also add to the JBoss7 entity a policy, the service restarter.

More policies

- Optimise to minimise latency to the users:
entities are moved to locations close to the users on the network (“follow the sun”)
- Optimise to minimise costs:
entities are moved to locations offering the lowest prices (“follow the moon”)
- Policies can be based on anything measurable!

The Foundation of Brooklyn

Deployment and wiring Runtime management



Autonomic computing

Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users [...] The system makes decisions on its own, using high-level policies; it will constantly check and optimize its status and automatically adapt itself to changing conditions. An autonomic computing framework is composed of autonomic components (AC) interacting with each other [...] with sensors (for self-monitoring), effectors (for self-adjustment), knowledge and planner/adaptor for exploiting policies based on self- and environment awareness.

https://en.wikipedia.org/wiki/Autonomic_computing

Autonomic computing is a key principle underlying Brooklyn. However you can use - and even develop - Brooklyn without needing detailed knowledge about autonomics. As a case in point, I know very little about autonomic computing! I have prepared a guide to help autonomic newbies to get started...

The bluffer's guide to autonomic computing in Brooklyn

- The autonomic components are the entities
- Entities have:
 - Sensors, which provide data to the external world
 - Effectors, which cause the entity to change in some way
- Sensor data drives policies;
policies drive effectors to make changes;
a continually-adapting feedback loop
- Management can happen locally at the entity;
or be escalated up the tree to be managed there

The status of Apache Brooklyn

Status update

- Version 0.7.0-M1 last before incubation
- Entered Apache Incubator in May 2014
- Version 0.7.0-M2-incubating expected imminently
- Final 0.7.0 expected by year end
- Code is still in rapid development pre-1.0

Status update

- 6 committers/PPMC members
- 10 mentors
- A number of additional developers
- A number of commercial customers (via Cloudsoft)
- A number of academic users
- but we are still a small community

How to help

- We need a bigger and more diverse community
- ...so please join us!
- Download and run, try out deployments
- Share your experiences on the mailing list
- Bug reports, code contributions, documentation contributions
- **Tell us how to make it better!**

Where to find us

- Official website: <https://brooklyn.incubator.apache.org>
- Mailing list:
dev-subscribe@brooklyn.incubator.apache.org
https://mail-archives.apache.org/mod_mbox/incubator-brooklyn-dev/
- Source code:
<https://github.com/apache/incubator-brooklyn>
or <https://git-wip-us.apache.org/repos/asf?p=incubator-brooklyn.git>
- IRC channel: #brooklyncentral on Freenode

Questions

We are hiring!

- Cloudsoft are looking for great software engineers
- To work on Apache Brooklyn and other open source projects, and with our commercial clients who are putting it into production
- Brooklyn is written in Java with a JavaScript front-end, but Java/JavaScript experience not an issue - because we know that great software engineers can adapt and learn
- Location not an issue - we have a distributed team

jobs@cloudsoft.io

apache **brooklyn**

Stay tuned for:

Clocker:

**Migrating Complex Applications To Docker
With Apache Brooklyn**

Presented by Andrew Kennedy, Cloudsoft

Next presentation in this room

apache **brooklyn**

Thank you!

Richard Downer

richard@apache.org - richard@cloudsoft.io

Twitter: @FrontierTown