Table of contents	
oud Support Plugin	

# **Cloud Support Plugin - Reference Documentation**

**Authors:** Burt Beckwith

**Version:** 1.0.11

### **Table of Contents**

1 Introduction to the Cloud Support Plugin

**1.1** History

2 Getting Started

## 1 Introduction to the Cloud Support Plugin

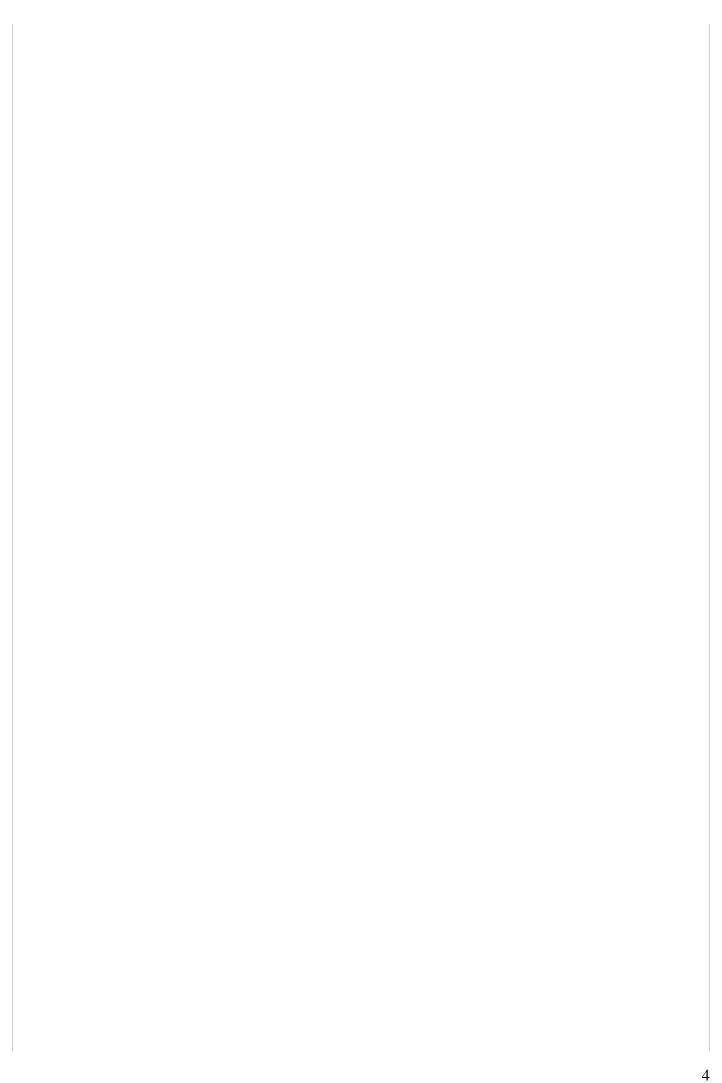
The Cloud Support plugin contains an abstract bean postprocessor base class that detects if any of Hibernate, Mongo, Redis, RabbitMQ, Searchable, and Memcached are installed and reconfigures them to connect to the cloud-provided services. Since gathering the connect information is cloud-specific, this is abstracted into methods that must be implemented in a concrete class.

This plugin isn't likely to be useful in general by itself, but rather as a dependency of other cloud plugins such as the Cloud Foundry plugin.

### 1.1 History

#### History

- May 21, 2012
  - 1.0.11 release
- May 21, 2012
  - 1.0.10 release
- March 6, 2012
  - 1.0.9 release
- December 24, 2011
  - 1.0.8 release
- October 7, 2011
  - 1.0.7 release
- October 5, 2011
  - 1.0.6 release
- October 4, 2011
  - 1.0.5 release
- October 4, 2011
  - 1.0.4 release
- October 2, 2011
  - 1.0.3 release
- October 2, 2011
  - 1.0.2 release
- October 2, 2011
  - 1.0.1 release
- September 30, 2011
  - Initial 1.0 release



## 2 Getting Started

The first step is to install the plugin, either with a dependency in BuildConfig.groovy

```
plugins {
compile ':cloud-support:1.0.11'
}
```

or with install-plugin

```
grails install-plugin cloud-support
```

The plugin provides two abstract base classes to help with configuring beans with connect information from the cloud environment. The first is AbstractCloudBeanPostprocessor. Subclass it in your project and implement the required methods (returning null from data methods where the service isn't supported or allocated). Then register it in your plugin's doWithSpring callback, e.g.

```
def doWithSpring = {
    myCloudBeanPostprocessor(MyCloudBeanPostprocessor)
}
```

The other is an abstract base class that configures Mongo (this can't be done in a bean post-processor, so it's done in doWithSpring). Like the AbstractCloudBeanPostprocessor subclass it mostly requires that you implement a method returning Mongo connect information. It doesn't need to be registered as a bean, but it should be invoked in doWithSpring, e.g.

```
def doWithSpring = {
    myCloudBeanPostprocessor(MyCloudBeanPostprocessor)

if (isCloudIsActive()) {
    new MyMongoBeanConfigurer().fixMongo(application)
    }
}
```