



Centralized Application Configuration with Spring and Apache ZooKeeper

By Ryan Gardner, Dealer.com

How do you configure your applications?

<http://b.socrative.com/login/student/>

config2gx

Who is Dealer.com?



We make software that car dealers use
to fulfill their digital marketing vision.

The Remote Configuration Project

Some ways we had configured our applications

- Hardcoded values in code
- Properties file – per environment or merged
- Host files for database
- JNDI context files

Motivating factors

- Developer Efficiency
 - Redeploying an application just to change a configuration is a drag
 - Having to edit N config files whenever a single application changed is a hassle
- Security Compliance
 - Limit access to production databases
 - Auditing and approval process for configuration changes
- Systems Engineering
 - Can't make certain changes without involving developers

Framework Development – Guns n Roses style

"Welcome to the jungle"

Thanks.

"We've got fun and games"

Cool.

"You're in the jungle"

We've established this

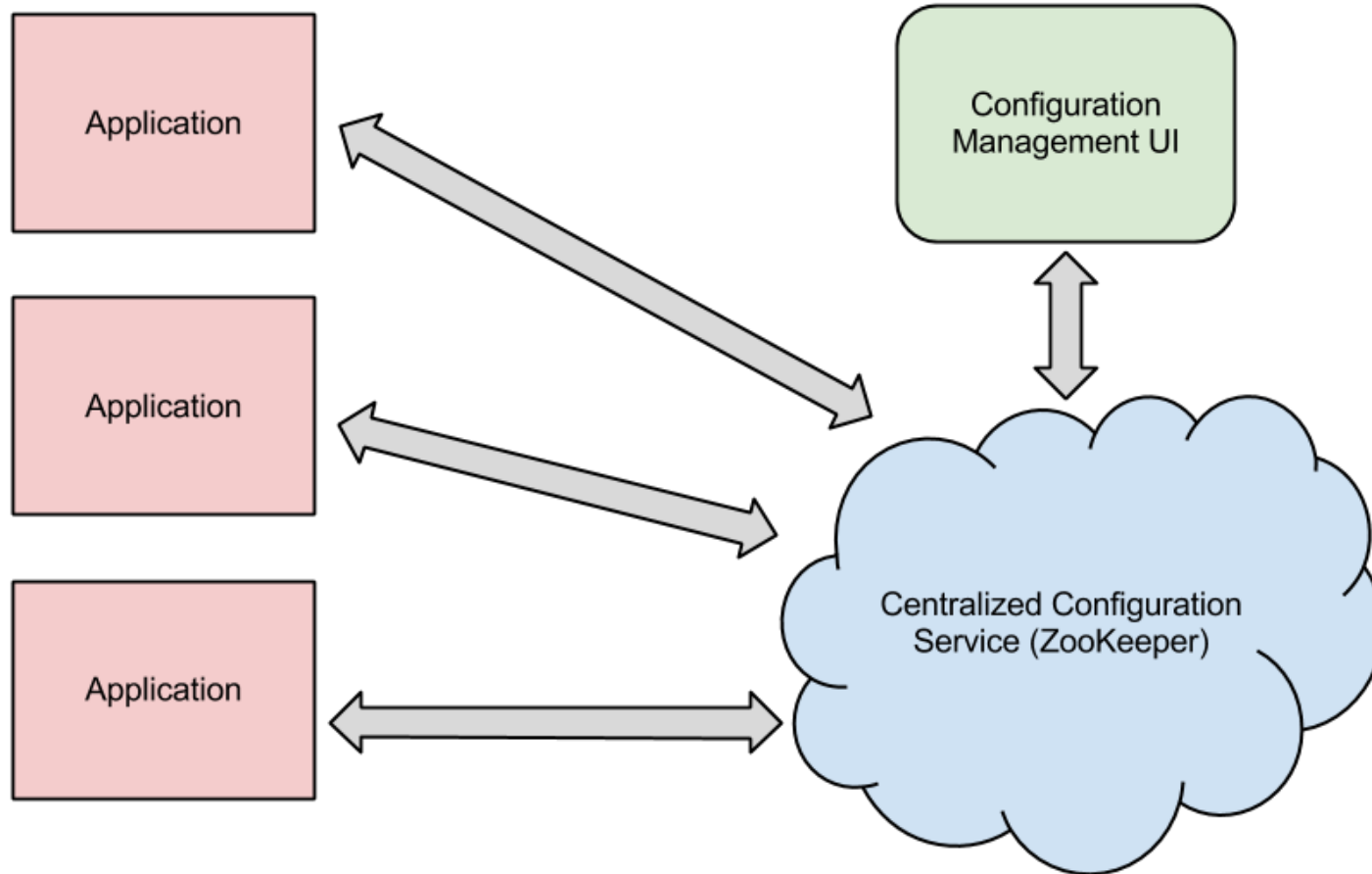
"You're gonna die!"

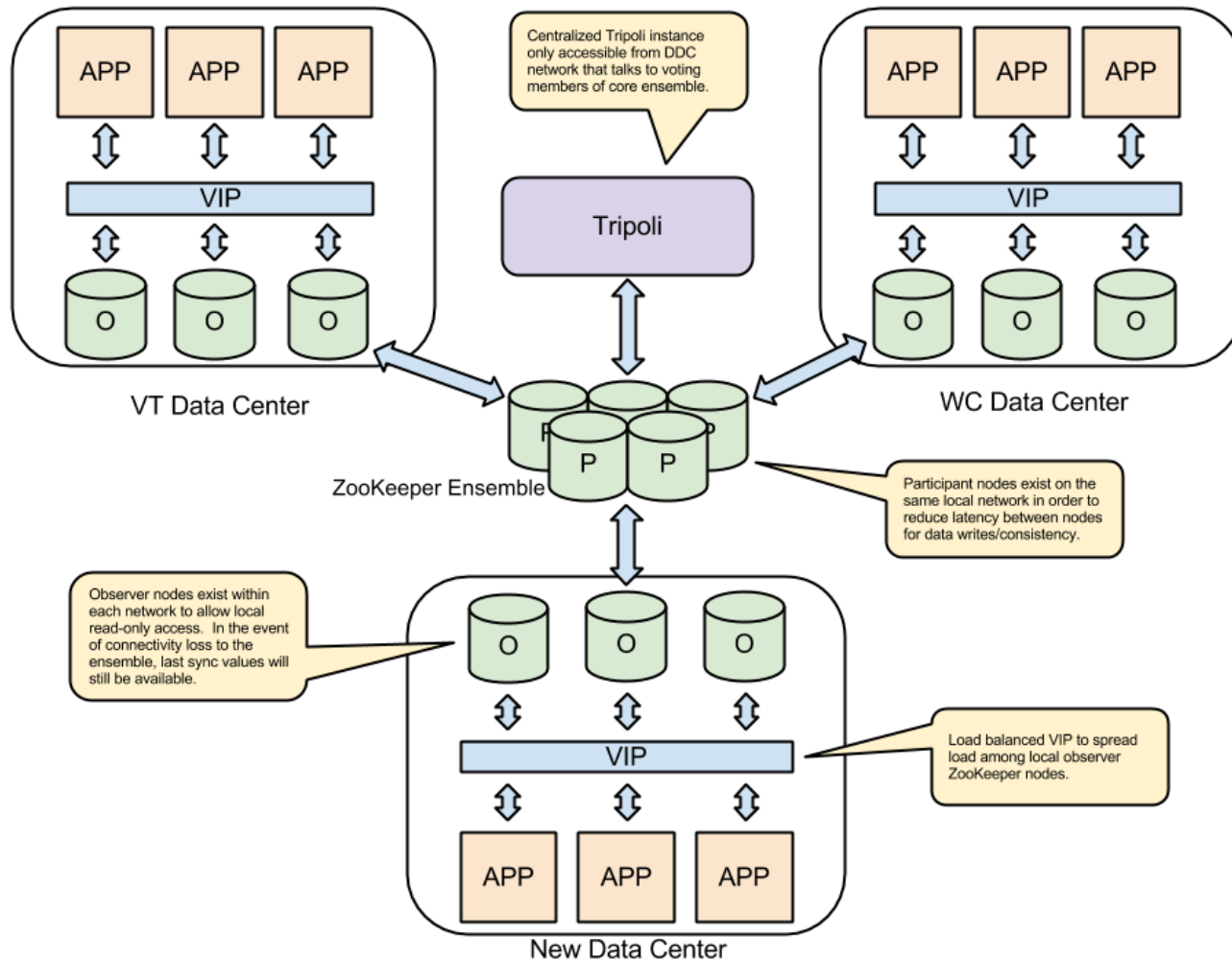
Wait what?

<https://twitter.com/OhNoSheTwitnt/status/469838190141255680>

High Level Overview

Three main components





Key Concepts in the Remote Configuration

Configuration

A set of properties or values necessary to create an object.

Examples:

Database configuration, FTP endpoint configuration,
HTTP Proxy Factory Bean Configuration,

Tripoli – editing a configuration



Editing a Configuration

/tripoli/configuration/ftp/call-swapping-akamai?type=com.dealer.config.provider.ft...

Tripoli

Changes

Tools

Jenkins

Logged in as **Ryan Gardner** (Logout)

call-swapping-akamai

ftp: com.dealer.config.provider.ftp.SpringIntegrationDefaultSftpSessionFactoryConfig

1 application has bindings affected by: /configs/ftp/dev/call-swapping-akamai

call-swapping-akamai

dev

vt

wc

qa

vt

wc

beta

vt

wc

live

vt

wc

+ Add Tag

Creator *

com.dealer.config.provider.ftp.SpringIntegrationDefaultSftpSessionFactory...

Host *

adtrack-static.

Port

22

Private Key *

.....

Private Key Passphrase

User Id *

sshacs

Validate

Request

Separation of environments

- We edit our configurations in one spot, but applications are only able to retrieve configurations for the environment they are running in
- Secrets – such as passwords or encryption keys – are encrypted with an environment-specific key
 - Tripoli has all the keys, each environment will only have a key specific to it

Configuration inheritance – avoiding copy & paste

By Environment

- Configuration can be set at a global level and overridden at each environment

By Path

- Nodes added with /'s in the name will inherit data from nodes above them.
 - `/remoting/core-services/UserLocator` inherits values from `/remoting/core-services`

Key Concepts in the Remote Configuration

Bindings

Identifying which configurations an application uses, and what the application wants to call them

Example: An application that needs to talk to a certain database, look up users from a remote service, and send data to a remote SFTP site would have bindings such as:

`jdbc/user-database` is bound to the configuration called `user-database-config`

Tripoli – editing bindings

The screenshot shows the Tripoli web application interface. The browser address bar indicates the URL is `/tripoli/bindings/catalog-services`. The application header shows the user is logged in as Ryan Gardner. The main heading is 'Application: catalog-services'. On the left, a sidebar lists environments: dev, vt, wc, qa, vt, wc, beta, vt, wc, live, vt, wc. The main content area is divided into three columns: Name, Configuration, and Tags. The 'Name' column lists three bindings: 'jdbc/dataSource', 'mongo/mongoDataSource', and 'properties/properties'. The 'Configuration' column shows dropdown menus for each binding, with values 'catalog-services-rw-tomcat', 'catalog-services-mongo', and 'catalog-services-properties' respectively. The 'Tags' column shows a text input field for each binding, all containing the value 'tags'. At the bottom, there is a section for 'Add a new binding:' with a form containing a 'category...' dropdown, a 'name' text input, a 'configuration' dropdown, and a 'tags' text input. At the bottom right, there are buttons for 'Validate' and 'Request'.

Editing Bindings

`/tripoli/bindings/catalog-services`

Logged in as Ryan Gardner (Logout)

Application: catalog-services

global

- dev
- vt
- wc
- qa
- vt
- wc
- beta
- vt
- wc
- live
- vt
- wc

Name

- jdbc/dataSource
- mongo/mongoDataSource
- properties/properties

Configuration

- catalog-services-rw-tomcat
- catalog-services-mongo
- catalog-services-properties

Tags

- tags
- tags
- tags

Add a new binding:

category... / name configuration tags

✓ Validate Request

Creating objects, not properties*

- Configure once – use everywhere
- Avoid having to copy-and-paste boilerplate setup code

* *properties are supported too*

Behind the Scenes

Apache Zookeeper

- Hierarchical data registers
- Designed for high-throughput, low-latency, highly-available
- Nodes in zookeeper are called “znodes”
 - Each path can stored data
- Designed for storing *small amounts of data* in the znodes (KB, not MB)
- For more info:
 - <https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription>

Where do we store this data?

- Versioned configuration in ZooKeeper as JSON
- In ZooKeeper znodes:
 - `/bindings/<binding name>`overrides:
 - `/bindings/<habitat>/<datacenter>/<binding name>`
 - `/configurations/<configuration name>/`

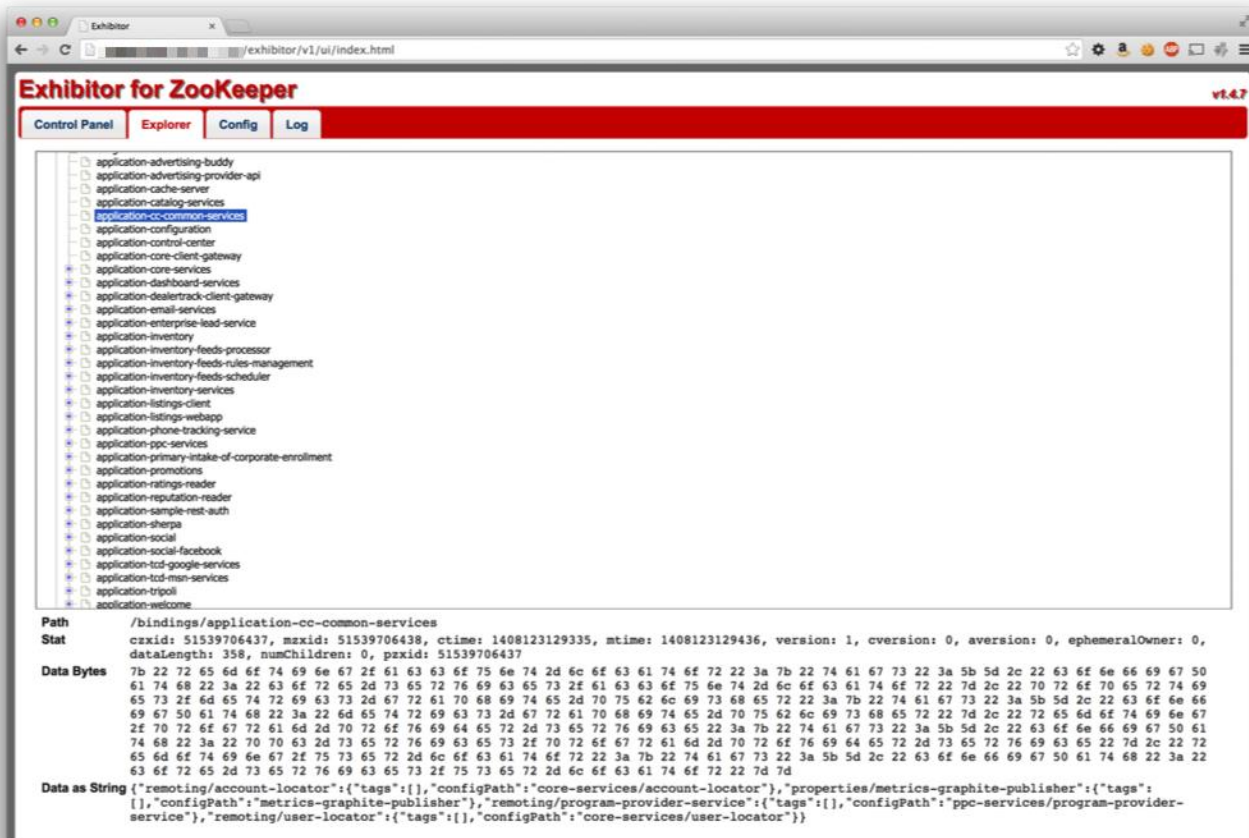


Talking to ZooKeeper

- Use Curator framework
- We use ACLs in ZooKeeper ensure apps can't read data for other environments
- We use a SASL config file on the machines to provide the ZooKeeper credentials



Exhibitor – makes managing ZooKeeper easier



The screenshot shows the Exhibitor for ZooKeeper web interface in a browser window. The interface has a red header bar with the title "Exhibitor for ZooKeeper" and version "v1.4.7". Below the header is a navigation bar with tabs: "Control Panel", "Explorer", "Config", and "Log". The "Explorer" tab is selected, showing a tree view of ZooKeeper nodes. The tree view lists various application services, with "application-cc-common-services" selected. Below the tree view, the "Data as String" section displays the raw ZooKeeper data for the selected node, including metadata like "Path", "Stat", and "Data Bytes".

Exhibitor for ZooKeeper v1.4.7

Control Panel Explorer Config Log

- application-advertising-buddy
- application-advertising-provider-api
- application-cache-server
- application-catalog-services
- application-cc-common-services**
- application-configuration
- application-control-center
- application-core-client-gateway
- application-core-services
- application-dashboard-services
- application-dealertrack-client-gateway
- application-email-services
- application-enterprise-lead-service
- application-inventory
- application-inventory-feeds-processor
- application-inventory-feeds-rules-management
- application-inventory-feeds-scheduler
- application-inventory-services
- application-listings-client
- application-listings-webapp
- application-phone-tracking-service
- application-ppc-services
- application-primary-intake-of-corporate-enrollment
- application-promotions
- application-ratings-reader
- application-reputation-reader
- application-sample-rest-auth
- application-sherpa
- application-social
- application-social-facebook
- application-tcd-google-services
- application-tcd-man-services
- application-tripoli
- application-welcome

Path /bindings/application-cc-common-services

Stat czxid: 51539706437, mxid: 51539706438, ctime: 1408123129335, mtime: 1408123129436, version: 1, cversion: 0, aversion: 0, ephemeralOwner: 0, dataLength: 358, numChildren: 0, pzxid: 51539706437

Data Bytes

```
7b 22 72 65 6d 6f 74 69 6e 67 2f 61 63 63 6f 75 6e 74 2d 6c 6f 63 61 74 6f 72 22 3a 7b 22 74 61 67 73 22 3a 5b 5d 2c 22 63 6f 6e 66 69 67 50
61 74 68 22 3a 22 63 6f 72 65 2d 73 65 72 76 69 63 65 73 2f 61 63 63 6f 75 6e 74 2d 6c 6f 63 61 74 6f 72 22 7d 2c 22 70 72 6f 70 65 72 74 69
65 73 2f 6d 65 74 72 69 63 73 2d 67 72 61 70 68 69 74 65 2d 70 75 62 6c 69 73 68 65 72 22 3a 7b 22 74 61 67 73 22 3a 5b 5d 2c 22 63 6f 6e 66
69 67 50 61 74 68 22 3a 22 6d 65 74 72 69 63 73 2d 67 72 61 70 68 69 74 65 2d 70 75 62 6c 69 73 68 65 72 22 7d 2c 22 72 65 6d 6f 74 69 6e 67
2f 70 72 6f 67 72 61 6d 2d 70 72 6f 69 64 65 72 2d 73 65 72 76 69 63 65 22 3a 7b 22 74 61 67 73 22 3a 5b 5d 2c 22 63 6f 6e 66 69 67 50 61
74 68 22 3a 22 70 70 63 73 65 72 76 69 63 65 73 2f 70 72 6f 72 61 6d 2d 70 72 6f 76 69 64 65 72 2d 73 65 72 76 69 63 65 22 7d 2c 22 72
65 6d 6f 74 69 6e 67 2f 75 73 65 72 2d 6c 6f 63 61 74 6f 72 22 3a 7b 22 74 61 67 73 22 3a 5b 5d 2c 22 63 6f 6e 66 69 67 50 61 74 68 22 3a 22
63 6f 72 65 2d 73 65 72 76 69 63 65 73 2f 75 73 65 72 2d 6c 6f 63 61 74 6f 72 22 7d
```

Data as String {"remoting/account-locator":{"tags":[],"configPath":"core-services/account-locator"},"properties/metrics-graphite-publisher":{"tags":
[],"configPath":"metrics-graphite-publisher"},"remoting/program-provider-service":{"tags":[],"configPath":"ppc-services/program-provider-
service"},"remoting/user-locator":{"tags":[],"configPath":"core-services/user-locator"}}

How do the objects get created?

- Two classes for each remotely-configurable object, the config and the creator
- Configs use bean-validation annotations and a special annotation on the config-field to explain what the config field does.
 - This populates the tool-tips in the browser window and is used to ensure that only valid entries are put into the fields

An example config class

```
@ConfigCategory("ftp")
public class SpringIntegrationSftpSessionFactoryConfig extends Config {

    @Required
    @ConfigField(description = "The host name of the SFTP server.")
    private String host;

    @Port
    @ConfigField(description = "The port of the SFTP server. Defaults to 22.")
    private Integer port;

    ...
}
```


A config class (continued)

@Required

@Password

@ConfigField(description = "The private key used to establish the SFTP connection.")

private ConfigPassword **privateKey**;

@Password

@ConfigField(description = "The passphrase for the private key. Defaults to empty string.")

private ConfigPassword **privateKeyPassphrase**;

Creators take the config and return an object

```
public class ExampleObjectCreator extends
    ObjectCreator<SomeConfig,ExampleObject> {

    @Override
    public ExampleObject create(SomeConfig) {
        // do whatever is needed to create the object
        return new ExampleObject();
    }
}
```

Kinds of creators we have made

- Database connection (various connection pools)
- Mongo connection pools
- RPC remoting proxies (Spring HTTP Invoker, etc)
- REST resources
- Redis connections
- Properties / System properties
- FTP and SFTP connections
- Executor services
- RabbitMQ
- SOLR
- ElasticSearch
- ... more

How do apps use this?

First pass – XML namespace parser

```
<beans ... xmlns:remote-config="http://www.dealer.com/schema/remote-config"  
...>  
  <remote-config:lookup id="dataSource" name="jdbc/my-datasource" />  
  
  <remote-config:remote-config-property-source id="myProps"  
    name="properties/my-props" />  
</beans>
```

Second pass – Auto-config with XML

```
<beans ... xmlns:remote-config="http://www.dealer.com/schema/remote-config"
...>
...

    <remote-config:auto-create />
    ...
</beans>
```

Third pass - @EnableRemoteConfig

@EnableRemoteConfig

```
public class ApplicationConfig {  
    // insert tweetable app here.  
}
```

Accessing properties

Integrating remote properties into spring

- We create a `PropertySource`
- And we create a `PropertySourcesPlaceholderConfigurer`

Using properties via @Value

@Bean

```
public SomeBean someBean (@Value("${some.value}") someValue) {  
    return new SomeBean(someValue)  
}
```

...

```
@Value("${some.remote.property.value}")  
private String someValue;
```

Deeper dive – demo &
look at some of the code

Future plans & extensions for this

Questions?

Learn More. Stay Connected



Tweet: “#s2gx talk about zookeeper blew my mind!
Thanks @ryebrye and @springcentral”



@springcentral



| spring.io/video