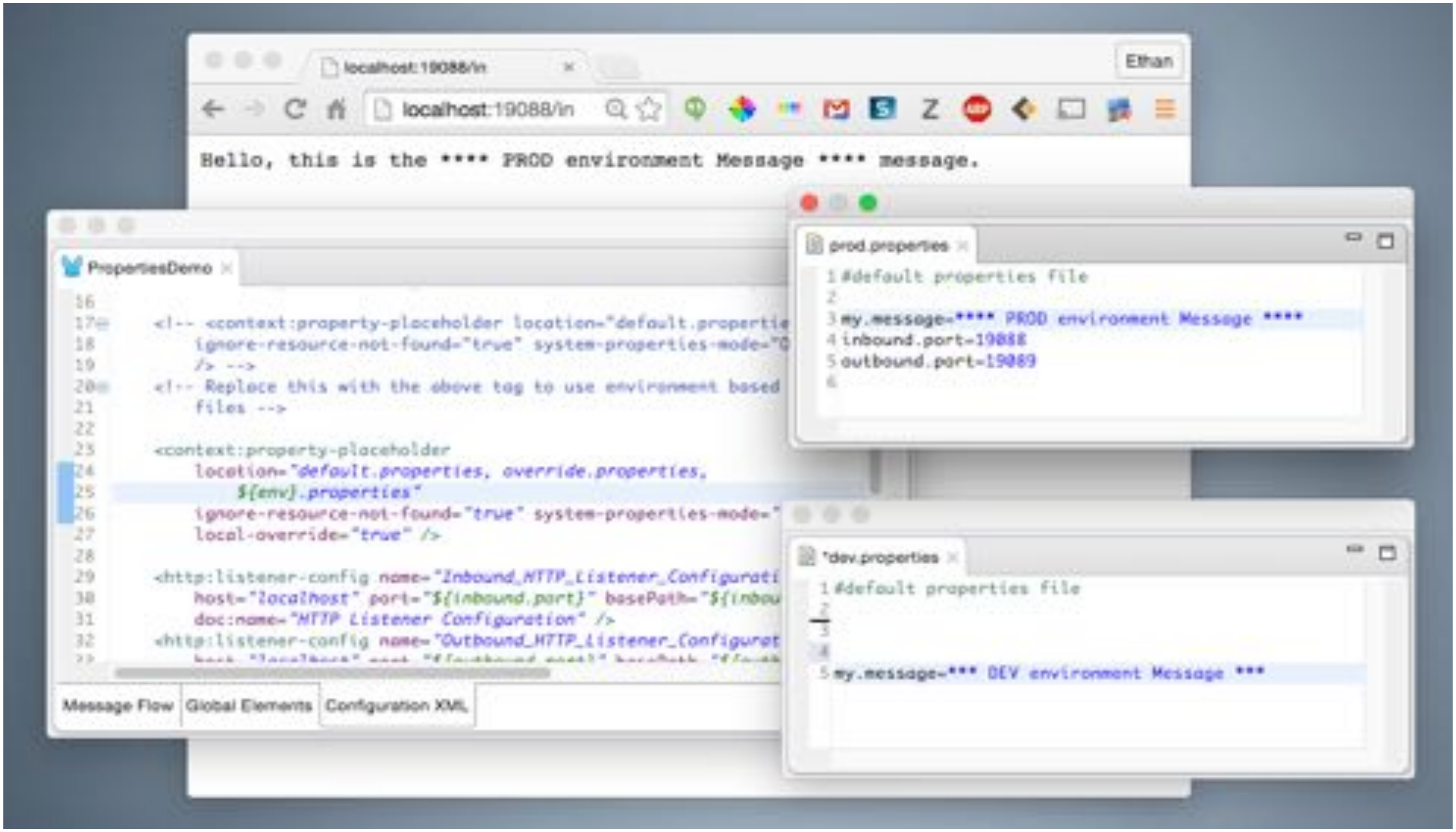
Module 4: Using Properties to Migrate Applications

Goals



```
Hello, this is the **** PROD environment Message **** message.
```

```xml
<!-- <context:property-placeholder location="default.propertie
    ignore-resource-not-found="true" system-properties-mode="O
    /> -->
<!-- Replace this with the above tag to use environment based
    files -->

<context:property-placeholder
    location="default.properties, override.properties,
        ${env}.properties"
    ignore-resource-not-found="true" system-properties-mode="
    local-override="true" />

<http:listener-config name="Inbound_HTTP_Listener_Configurati
    host="localhost" port="${inbound.port}" basePath="${inbou
    doc:name="HTTP Listener Configuration" />
<http:listener-config name="Outbound_HTTP_Listener_Configurat
```

prod.properties

```
#default properties file

my.message=**** PROD environment Message ****
inbound.port=19088
outbound.port=19089
```

*dev.properties

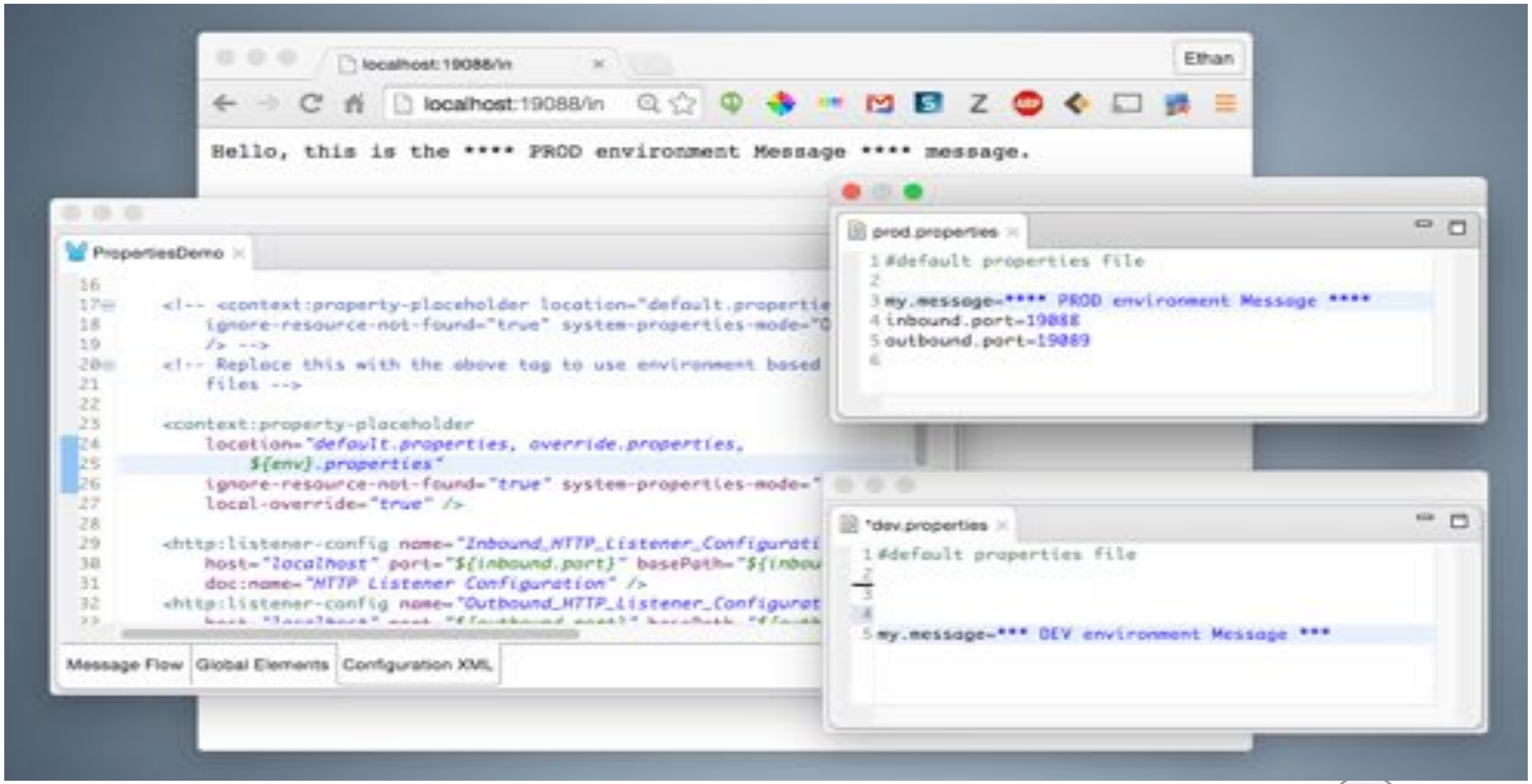
```
#default properties file



my.message=*** DEV environment Message ***
```

Objectives:

- Set property placeholder values in applications
- Override default properties in deployed applications
- Configure and use environment specific properties files



```
Hello, this is the **** PROD environment Message **** message.
```

```
    <!-- <context:property-placeholder location="default.propertie
        ignore-resource-not-found="true" system-properties-mode="O
        /> -->
    <!-- Replace this with the above tag to use environment based
        files -->

    <context:property-placeholder
        location="default.properties, override.properties,
            ${env}.properties"
        ignore-resource-not-found="true" system-properties-mode="
        local-override="true" />

    <http:listener-config name="Inbound_HTTP_Listener_Configurati
        host="localhost" port="${inbound.port}" basePath="${inbou
        doc:name="HTTP Listener Configuration" />
    <http:listener-config name="Outbound_HTTP_Listener_Configurat
```

```
#default properties file

my.message=**** PROD environment Message ****
inbound.port=19088
outbound.port=19089
```

```
#default properties file



my.message=*** DEV environment Message ***
```

Topics

- Setting Properties:
  - Properties, placeholders and properties files
  - Overriding property files
- Environment Variables

Setting Application Properties

Intro to Properties

- When developers build applications they need to provide:
  - Server addresses and ports
  - Access credentials
  - Database or Queue names
  - Other environment-specific bits of data
  - There are different environments for Dev, QA, Staging, Prod:
    - All with different server names, access credentials, etc.
  - For obvious reasons, it’s NOT a good idea to give developers PROD credentials
- Solution:
  - Make the application itself independent of the environment
  - Configure all environment-dependent values outside of the app

Property Placeholders

- Mule, which is based on Spring, allows the use of placing *tokens* inside ${} property placeholders. For example:
  - <inbound-endpoint address="**${*my.inbound.address*}**" />
- Developers should use ${} placeholders when possible:
  - Makes applications more flexible and adaptable
  - Allows tweaking once the application is in Production

```
<mule …>
    <context:property-placeholder
location="/jms.properties" />
    <jms:connector brokerUrl=
"${jms.server}:${jms.port}" />

    <flow …>
        <jms:inbound-endpoint
queue="${jms.inbound.queue}" />
        …
    </flow>
</mule>
```

```
#jms.properties

jms.server=mycompany.com
jms.port=7777
jms.inbound.queue=jms.orders
```

Properties and Environment Variables

- Developers: should not hardcode dynamic values in applications
  - Use ${placeholders} instead

```
#[ ' The server name is ${app.servername} ' ]
```

- Options:
  - Application level Environment variables
  - Domain variables
  - Property Placeholder files

- Environment Variables
  - Can be edited per domain
  - Take precedence over Property Placeholder files

Properties files

- Properties can be read straight from properties files:

```
<context:property-placeholder location="server.properties" />
```

- Mule searches the application's classpath to find these files:
  - *$MULE_HOME/conf*
  - *$MULE_HOME/lib/shared/**
  - *$MULE_HOME/apps/<app>/classes*
  - *$MULE_HOME/apps/<app>/lib/*.jar*
- Multiple properties files can be imported:

```
<context:property-placeholder location="dev.properties,
        prod.properties, *.props"
        ignore-resource-not-found="true" />
```

Property Placeholder Locations (Overview)

1. Store application property placeholders in an external location, outside the $MULE_HOME
   - Hardcode the external location
   - Use an environment variable to switch the external location between environments
     - Set in wrapper.conf or on mule server startup with –M-D option
2. Store application property placeholders inside the $MULE_HOME
   - Store inside the application's deployment folder
   - Store anywhere in the Mule server's CLASSPATH
     - $MULE_HOME/apps or $MULE_HOME/conf

Option 1: Store configuration files outside $MULE_HOME

- Put all configuration property placeholder files outside the $MULE_HOME location
  - This keeps properties stable after Mule server or application upgrades
  - Also supports lifecycle governance
    - PCI / Sarbanes-Oxley

```
/opt
   /mule_3.7
      /apps
         /appX
      /conf
         wrapper.conf
   /mule_conf
      appX.properties
      appY.properties
```

- Specify the external conf file location
  - In a hard-coded external location

```
<context:property-placeholder
location="/opt/myCompany/mule/appX.properties" />
```

  - In a location specified by an Environment variable
    - ./mule -M-DMULE_CONF=C:\myCompany\mule

```
<context:property-placeholder
location="file://${MULE_CONF}/appX.properties" />
```

Option 2: Establish a hierarchy

- Override properties from outside an app:
  1. Put all configurable properties in a file and place it in *$MULE_HOME/conf/*
     - And NOT in the app

```
<context:property-placeholder location=”appX.properties" />
```

  2. Put default (or dev) properties in the app and override them from *$MULE_HOME/conf/*

```
<context:property-placeholder location="appX.properties, override.properties"
                                          ignore-resource-not-found="true"/>
```

  – And then set the values:

```
#appX.properties
jms.server = dev.server.com
jms.port = 61616
jms.inbound.queue = input
```

```
#override.properties
jms.server = prod.server.com
```

MuleSoft

Other ways of setting properties

- <global-property>
  - Within the same or another Mule configuration file:

```
<global-property name=“jms.server” value=“my.server.com” />
```

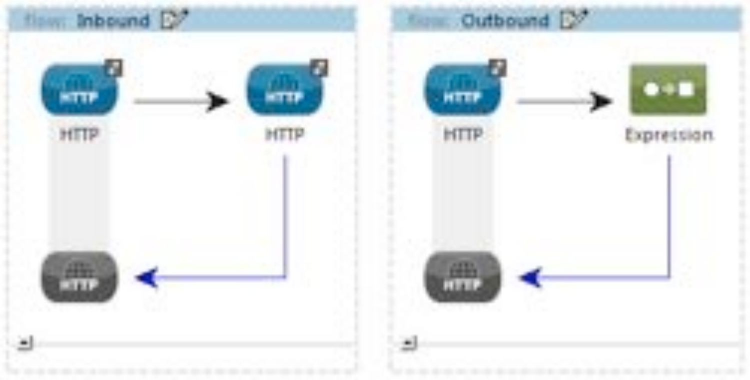
- System properties:
  - From the command line
    - mule –M-Djms.server=my.server.com –M-Denv=dev
  - From *$MULE_HOME/conf/wrapper.conf*
    - java.additional.19=jms.server=my.server.com
    - java.additional.20=-Denv=prod

- From *web.xml* (when embedding Mule in web container)

```
<context-param>
  <param-name>mule.serverId</param-name>
  <param-value>MyServer</param-value>
</context-param>
```

Walkthrough 4-1: Override Property Placeholders

- Deploy “propertiesDemo.zip” in one of your Mules
- This app has two flows:
  - Inbound flow calls Outbound flow
  - And returns a message



- Override properties in other properties files

Domains and Properties

- You can also add properties files to domains

- Developers can bundle applications into domains
  - Developers create domain projects to bundle applications
  - They all share the same global resources
  - Only some connectors are supported for domains
    - See the documentation

Steps to set the environment for a Mule Server

- If you do not hardcode the properties files location
- Operators start Mule server instance with environment variable(s) set (e.g. env=dev)
- One of these:
  - 1. Edit wrapper.conf

```
wrapper.java.additional.20=-Denv=dev
wrapper.java.additional.20=-DMULE_CONF=/opt/mule_conf
```

  - 2. Add startup Java environment variable

```
mule.bat -M-Denv=dev -M-DMULE_CONF=C:\mule_conf

./mule -M-Denv=prod -M-DMULE_CONF=/opt/mule_conf
```

MuleSoft

Steps to migrate between environments

- Set an Environment variable
  - Example env=dev
- Developers add Property Placeholders to configuration values
  - e.g. `location=myapp-${env}.properties`
- Operators add a myapp-dev.properties file to the deployment CLASSPATH
  - $MULE_HOME/conf/
  - $MULE_HOME/<<app>>/classes/
- Modify property placeholders as desired

Walkthrough 4-2: Migrating properties between environments

- Set an Environment variable env=dev or env=prod
- Set the properties file as ${env}.properties
- Verify environments can be switched

Refreshing property placeholder file changes

- Normally you need to restart the Mule server to load in property placeholder file changes
- Either:
  - Touch the application’s configuration xml file
  - Restart the server

- Developers can also include the Mule Module Requestor in a project flow to read in the file upon demand

https://github.com/mulesoft/mule-module-requester

MuleSoft

Summary

- Developers use Environment variables and Property Placeholders to reuse configuration values
- Operators can override these properties in other files
- A Mule Server can set server-wide properties
- Environment variables allow operators to migrate properties between Mule servers (environments)

MuleSoft®