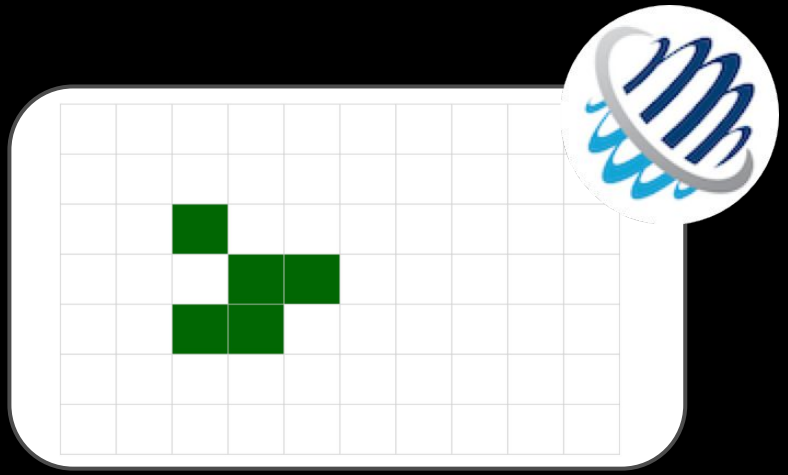


Virgo by Example

Florian Waibel, Markus Knauer

Survey Who has used…

… Virgo ?

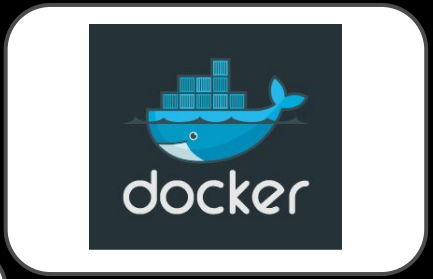
… WebSockets ?

… Docker ?

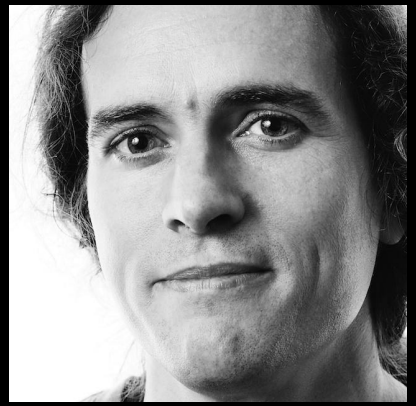
… Gradle ?

… git ?

RFC 6455
The WebSocket Protocol



Who we are



Florian

Markus

Eclipse Virgo by Example: Game of Life - Mozilla Firefox

Eclipse Virgo by Ex... ×

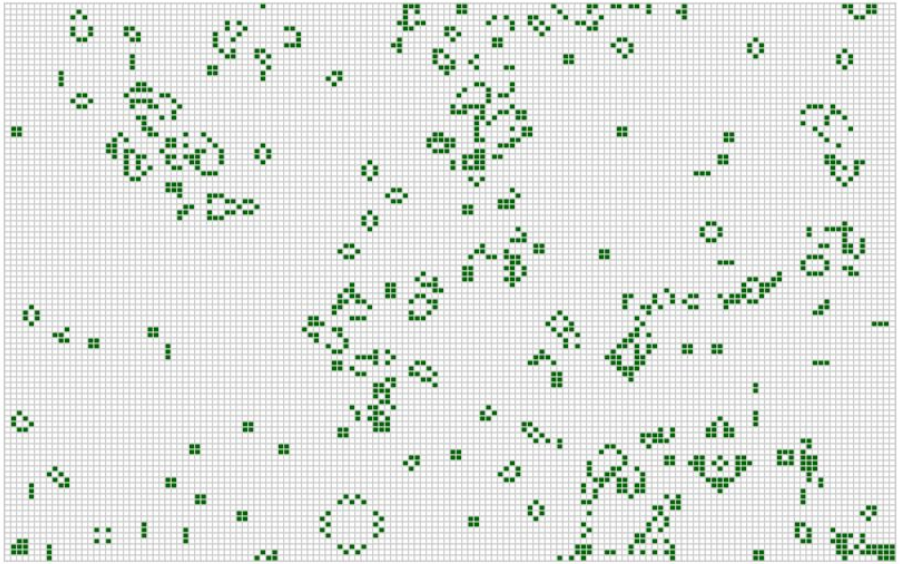
gol.eclipsesource.com/gol/board

Search

Eclipse Virgo by Example: Game of Life

This is a proof of concept build of Conway's game of life made using websockets.

The server calculates and generates the next board, and the client draws whatever the server tells it to.



Roadmap

1. Setup Workspace - Intro to Virgo Tooling
2. Embed JavaScript based “Game-of-Life” Jenova
3. Investigate game engine lifecycle
4. Add custom OSGi Command
5. Communicate via OSGi EventAdmin
6. Configure WebSocket
7. Build and Run with Docker

Installing Tutorial Prerequisites

Prerequisite 1: The IDE

1. Eclipse IDE for Java EE Developers
   a. Virgo Tooling
   b. Docker Tooling

➡ pre-packaged versions available!

Download prepackaged Eclipse

Go to http://gol.eclipsesource.com/downloads/
and download prepackaged Eclipse archive depending on OS

USB stick:

```
cp eclipse-jee-neon-M5-virgo-tutorial-macosx-cocoa-x86_64.tar.gz  ~/
```

Install Eclipse

Eclipse Neon M5 with

- Virgo Tooling (https://wiki.eclipse.org/Virgo/Tooling)
- Docker Tooling

USB stick:

```
unzip eclipse-jee-neon-M5-virgo-tutorial-win32-x86_64.zip
tar zvxf eclipse-jee-neon-M5-virgo-tutorial-macosx-cocoa-x86_64.tar.gz
```

Prerequisite 2: Custom Virgo Runtime

Go to http://gol.eclipsesource.com/downloads/
and download the Virgo Game-of-Life Runtime

USB stick:

```
cp virgo-gol-runtime.tar.gz ~/
```

Install Virgo Runtime

Eclipse Virgo 3.7.0.M02 with

- Spring 4.2.1.RELEASE
- Nashorn JavaScript engine
- JSON Mapper Jackson (https://github.com/FasterXML/jackson)

USB stick:

```
unzip virgo-gol-runtime.zip
tar xvfz virgo-gol-runtime.tar.gz
```

Verify Virgo Runtime Setup

- Grant access to OSGi console `${VIRGO_HOME}/repository/ext/osgi.console.properties`
- Start Virgo Runtime

  `${VIRGO_HOME}/bin/startup.sh`
- Go to Virgo Admin Console

  http://localhost:8080/admin/ (admin/admin)
- Connect to User Region via Telnet / SSH

  `telnet localhost 2501`

  `ssh -p 2502 admin@localhost` (pw: admin)

```
telnet.enabled=true
telnet.port=2501
telnet.host=localhost
ssh.enabled=true
ssh.port=2502
ssh.host=localhost
```

Prerequisite 3: The Git Repo

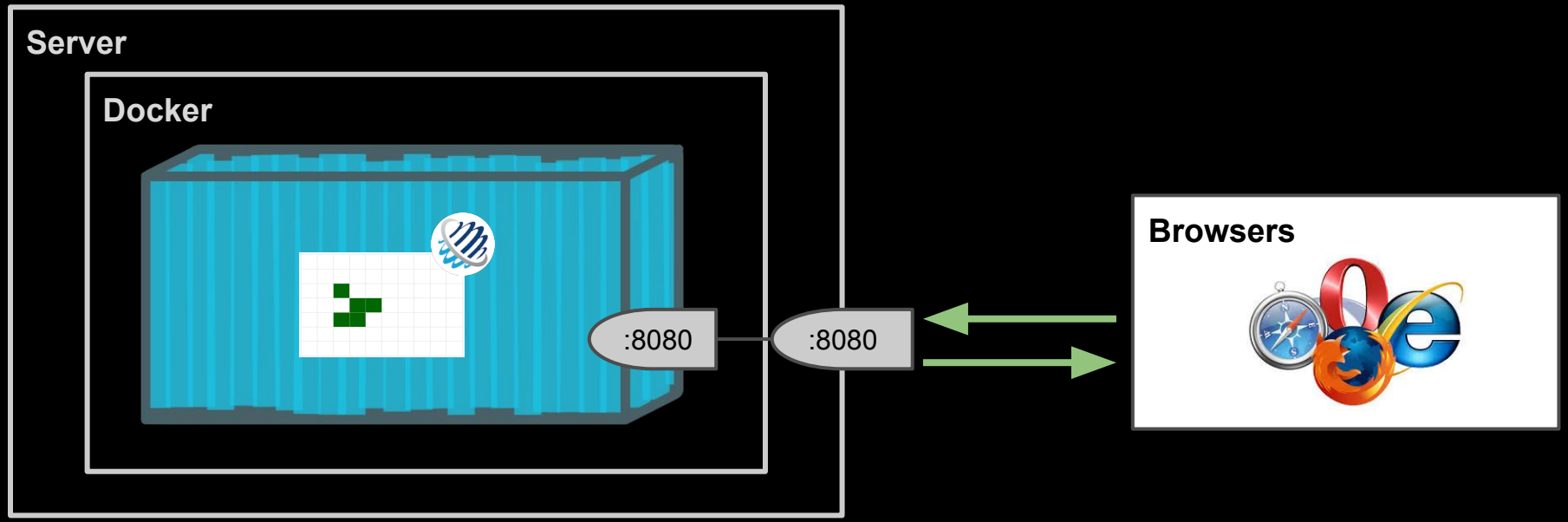

```
git clone https://github.
com/eclipsesource/virgo_game_of_life.git
```

USB Stick (Get local copy of the Git repository)

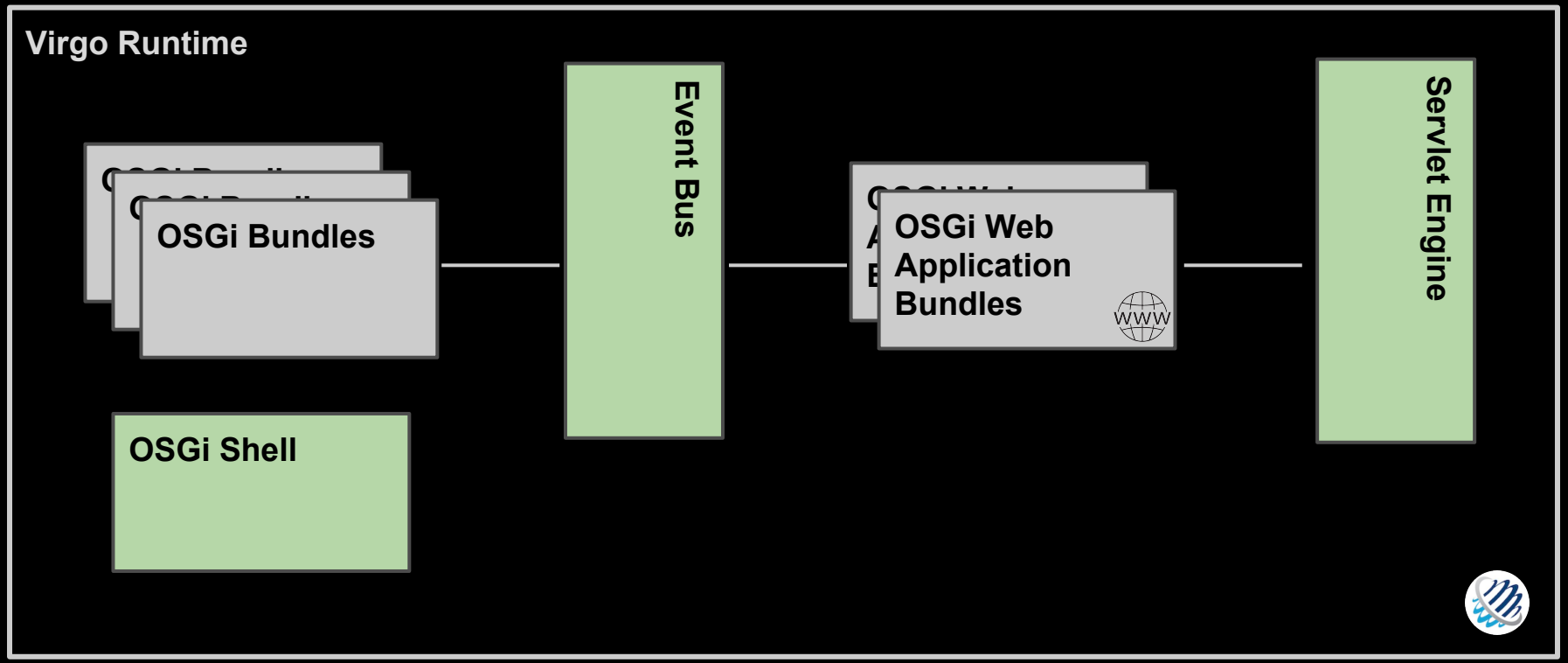
```
unzip virgo_game_of_life.zip -d ~/git/
```

10,000 Feet: Data Flow

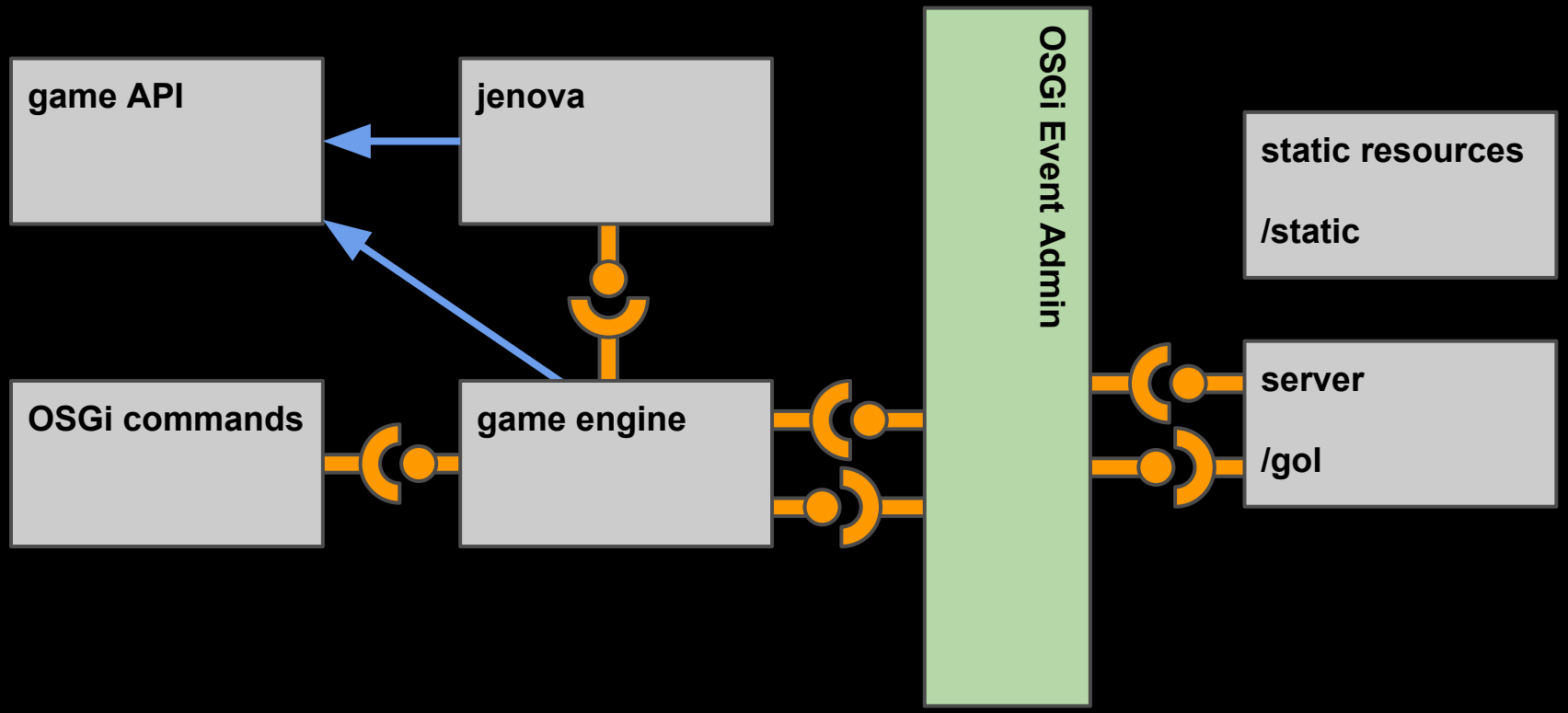
5,000 Feet: Docker Deployment



3,000 Feet: Backend



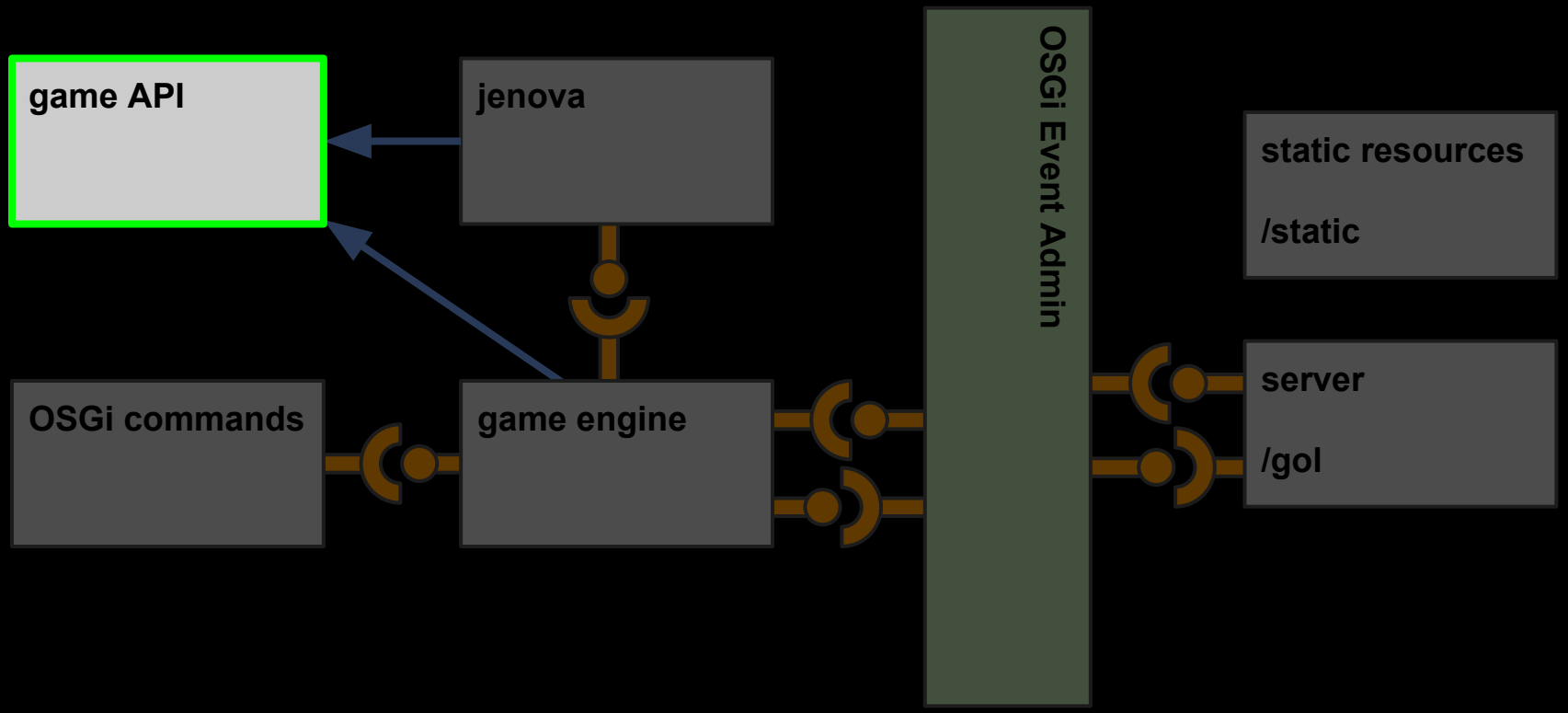
1,000 Feet: OSGi



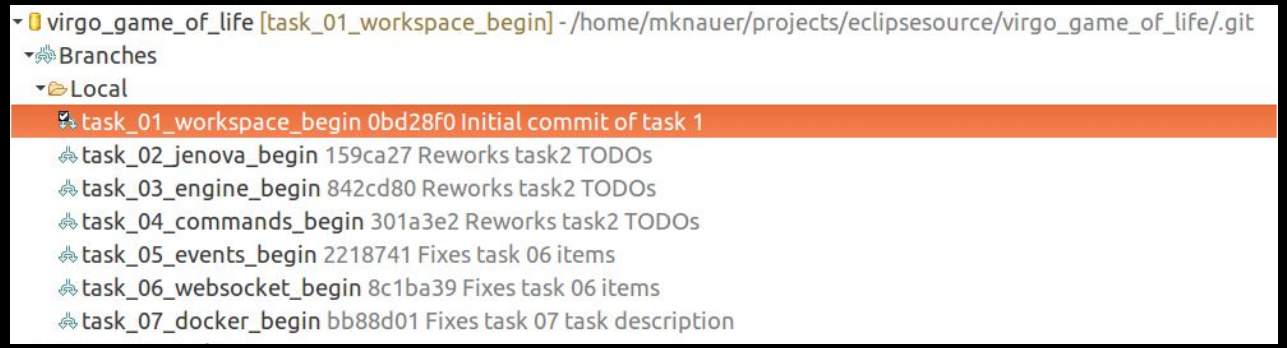
Ready, Steady, Vir...

Go!

Task 1: Workspace + API Bundle



Tutorial as Branches



During the Tutorial YOU do:

1. Try to solve the tasks
   (Hint: Look for **TODO** **`task_x.y`** in the code)
2. `git diff` **`task_x_<task_name>_final`**
3. `git checkout` **`task_x+1_<task_name>_begin`**

Start Game-of-Life Workspace

```
git checkout task_01_workspace_begin
```

USB Stick (Get local copy of Gradle dependencies)

Save your version of the cache

!

```
unzip gradle-cache.
zip -d ~/.gradle/
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

Import OSGi Bundle Projects

1. Switch to initial branch in your Git repo

```
$ cd virgo_game_of_life
$ git checkout task_01_workspace_begin
```

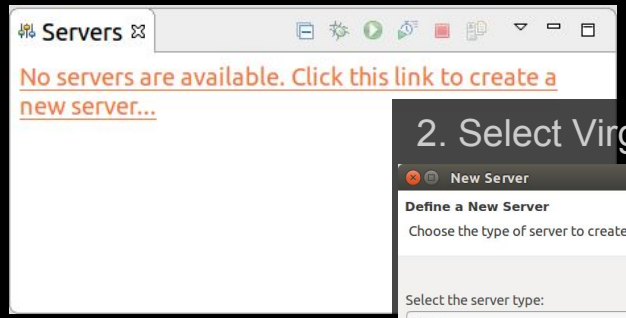
2. Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

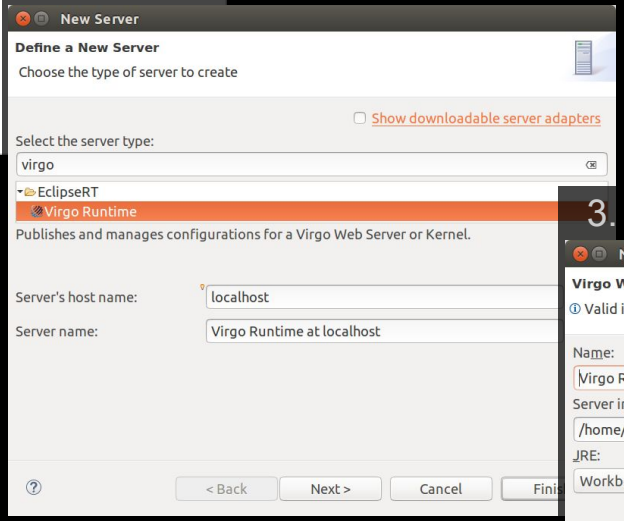
3. Start provided Eclipse with **new** workspace
4. Import... Gradle Project…

Create New Server Runtime

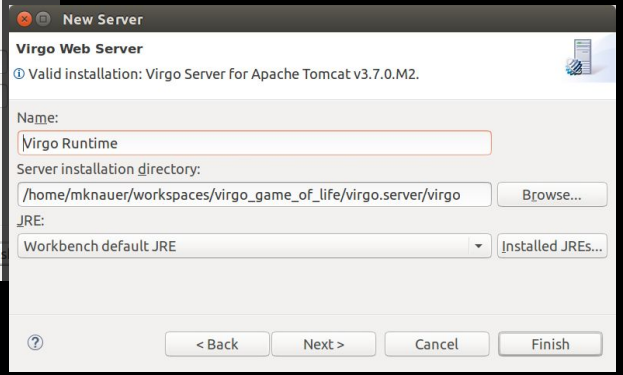
1. Open the Servers View



2. Select Virgo Runtime



3. Select path to Virgo Runtime



Configure Server Runtime

Drop your bundles onto server

Servers

Virgo Runtime at localhost [Stopped]

game-api [Synchronized]

Double-click on server, adjust publishing settings

Virgo Runtime at localhost

Overview

General Information

Specify the host name and other common settings.

Server name: Virgo Runtime at localhost

Host name: localhost

Runtime Environment: Virgo Runtime

Configuration path: /Servers/Virgo Runtime at localhost-config Browse...

Open launch configuration

Target Platform

Configure a PDE Target Platform definition for developing using PDE.

Reload the current Target Platform definition due to bundle changes or edit its content.

Redeploy Behavior

Configure server redeploy and refresh behavior on changes to project resources.

Define patterns for files that should be copied into the server without redeploying the bundle or application. Spring XML configuration files be handled separately.

*.html
*.xhtml
*.css
*.js
*.jspx

Add Delete Up Down

Restore default filename pattern

Publishing

Modify settings for publishing.

Never publish automatically

Automatically publish when resources change

Automatically publish after a build event

Publishing interval (in seconds): 5

Timeouts

Deployer Control

Artefact Deployment Order

Specify the deployment order of targeted bundles and PARs on server startup.

game-api

Up Down

Server Startup Configuration

Specify startup options. Changing a setting requires a server restart.

Tail deploy events into Console view

Start server with -clean option

-XX:MaxPermSize= 512m

Overview | Repository | Bundle Overview | Bundle Dependency Graph | Server Console

task 01.1 Fix template.mf

Project Explorer

- game-api [virgo_game_of_life task_01_workspace_begin]
  - src/main/java
  - JRE System Library [JavaSE-1.8]
  - src
  - build.gradle
  - template.mf
- Servers
- Virgo Runtime at localhost Server
- virgo_game_of_life [virgo_game_of_li

template: com.eclipsesource.examples.gol.api

Overview

General Information

This section describes general information about this bundle

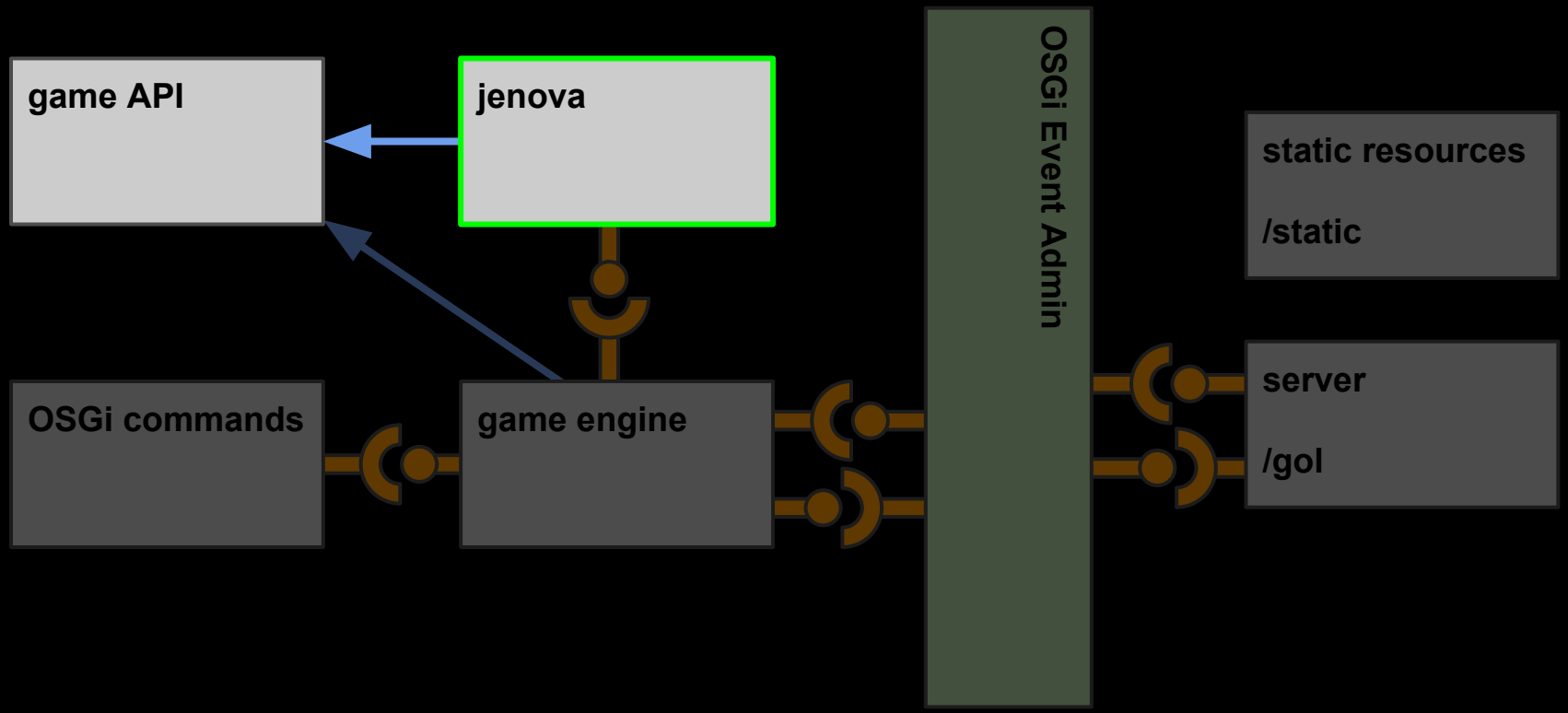
ID:	com.eclipsesource.examples.gol.api
Version:	0.1
Name:	TODO task 01.1 provide bundle name and check in OSGi console with headers command
Vendor:	
Activator:	Browse...

Verify Game-of-Life Workspace

```
osgi> ss api
"Framework is launched."
id      State      Bundle
126     ACTIVE   com.eclipsesource.examples.gol.api_0.1.0

osgi> headers 126
Bundle headers:
...
 Bundle-Name = Game of Life API
...
```

Task 2: Jenova - JavaScript



Start Jenova - Embedded JavaScript

```
git checkout task_02_jenova_begin
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

<lang:std />

Spring bean name

Java interface of the Spring bean

```
<lang:std id="jenova"
      engine="nashorn"
      script-interfaces="com.eclipsesource.examples.gol.api.GameOfLife">
   <lang:inline-script>
      <![CDATA[
// Conversion from Java int[][] to JavaScript [][]
// Function body of original Jenova JavaScript code
      ]]>
   </lang:inline-script>
</lang:std>
```

Convert the incoming Java int[][] to JavaScript Array and reuse the original function body of the Jenova Snippet

JSR-223 based mechanism for scripted beans, exposed through the `<lang:std />` element in XML. (backed by the `StandardScriptFactory`)

<osgi:service />

```
public interface GameOfLife {
    int[][] next(int[][] a);
}
```

Internal ID of the Spring bean backing the OSGi service

```
<osgi:service
    ref="jenova"
    interface="com.eclipsesource.examples.gol.api.GameOfLife" />
```

Interface of the registered OSGi service

Expose a referenced Spring bean as OSGi service with a given interface with the `<osgi:service />` element in XML

Task 2: Embedded JavaScript

02.1 add id 'jenova' and specify the matching Java interface

02.2 merge jenova.js here and verify result with JUnit test JenovaTest

02.3 expose JavaScript backed Jenova bean as OSGi service and verify result via OSGi console

Verify Green JUnit tests + Console

```
$ ./gradlew :jenova:test
```

```
osgi> services *GameOfLife
{com.eclipsesource.examples.gol.api.GameOfLife}={org.eclipse.gemini.blueprint.bean.
name=jenova, ..., Bundle-SymbolicName=com.eclipsesource.examples.gol.jenova, Bundle-
Version=0.1.0, service.id=251}
  "Registered by bundle:" com.eclipsesource.examples.gol.jenova_0.1.0 [127]
```

End Jenova - Embedded JavaScript

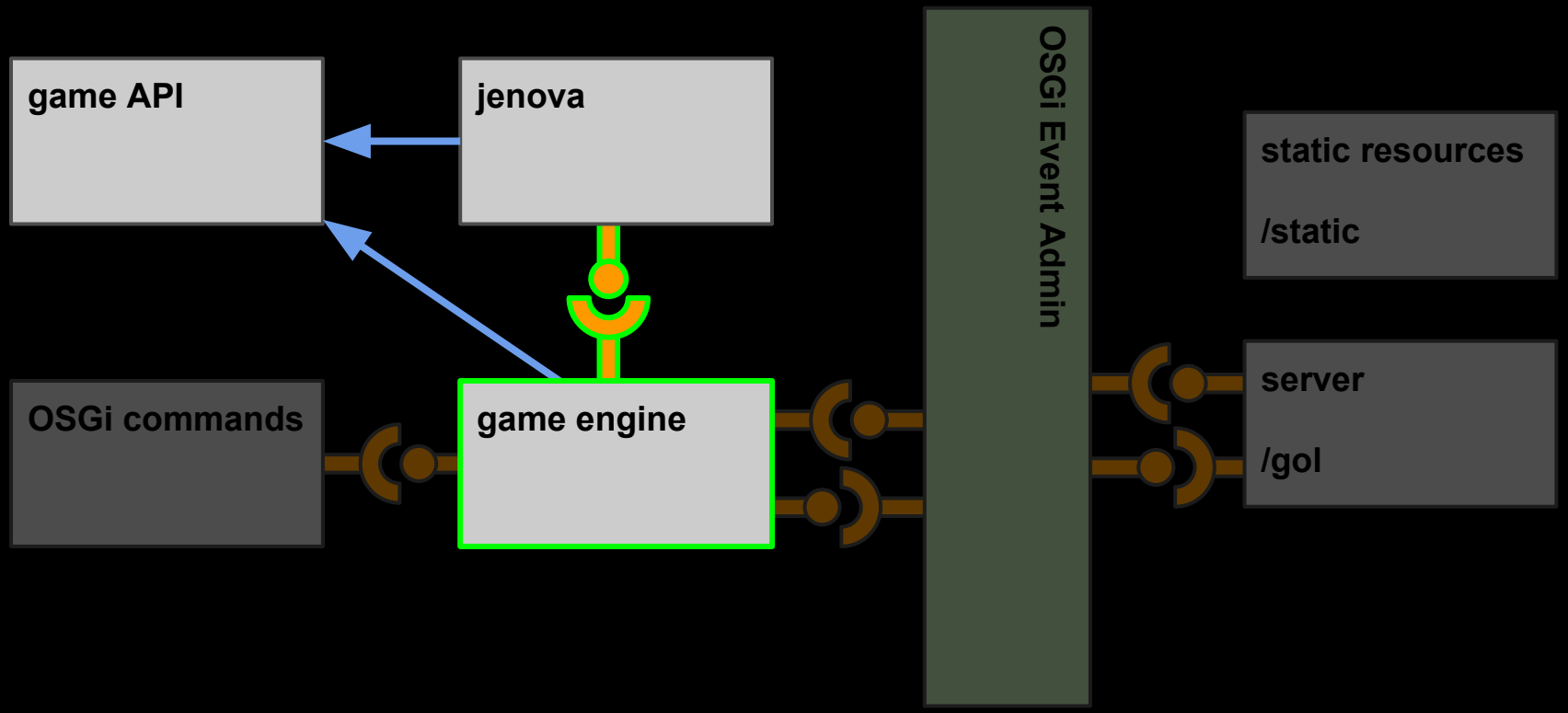
```
git diff task_02_jenova_final
```

Bonus Jenova - JavaScript

?

Consume the JavaScript snippet from the file system (i.e. not inlined in the XML)

Task 3: NanoService GameEngine



Start NanoService GameEngine

```
git checkout task_03_engine_begin
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

<osgi:reference />

```
public interface GameOfLife {
    int[][] next(int[][] a);
}
```

Internal ID of the Spring bean backed by the OSGi service

```
<osgi:reference
    id="jenova"
    interface="com.eclipsesource.examples.gol.api.GameOfLife" />
```

Interface of the referenced OSGi service

Publishes an OSGi reference as Spring bean named jenova with a given interface and the `<osgi:reference />` element in XML

Spring beans (Java + XML)

```
package c.e.e.gol.engine;

@Component("gameEngine")
public class DefaultGameEngine {

  @Autowired
  private GameOfLife gameOfLife;

  @PostConstruct
  public void init() {}

  @PreDestroy
  public void destroy() {}
}
```

Name of the Spring component

Inject GameOfLife bean

Spring bean lifecycle hooks

```
<context:component-scan
        base-package="c.e.e.gol.engine" />
```

All classes within the base package will be processed by Spring

Task 3: Nano service `GameEngine`

03.1 autowire `GameOfLife`

03.2 start bean post construction

03.3 calculate and store next generation of the board

03.4 shutdown bean pre destruction

03.5 enable component scan for bundle game engine

03.6 reference OSGi service `GameOfLife` as bean with id jenova

03.7 publish `GameEngine` as OSGi service

Verify NanoService GameEngine

```
$ tail -f ${VIRGO_HOME}/serviceability/logs/log.log
 -- Calculating next generation --
 -- Calculating next generation --
 -- Calculating next generation --
...
```

```
osgi> services *GameEngine
?
```

End NanoService GameEngine

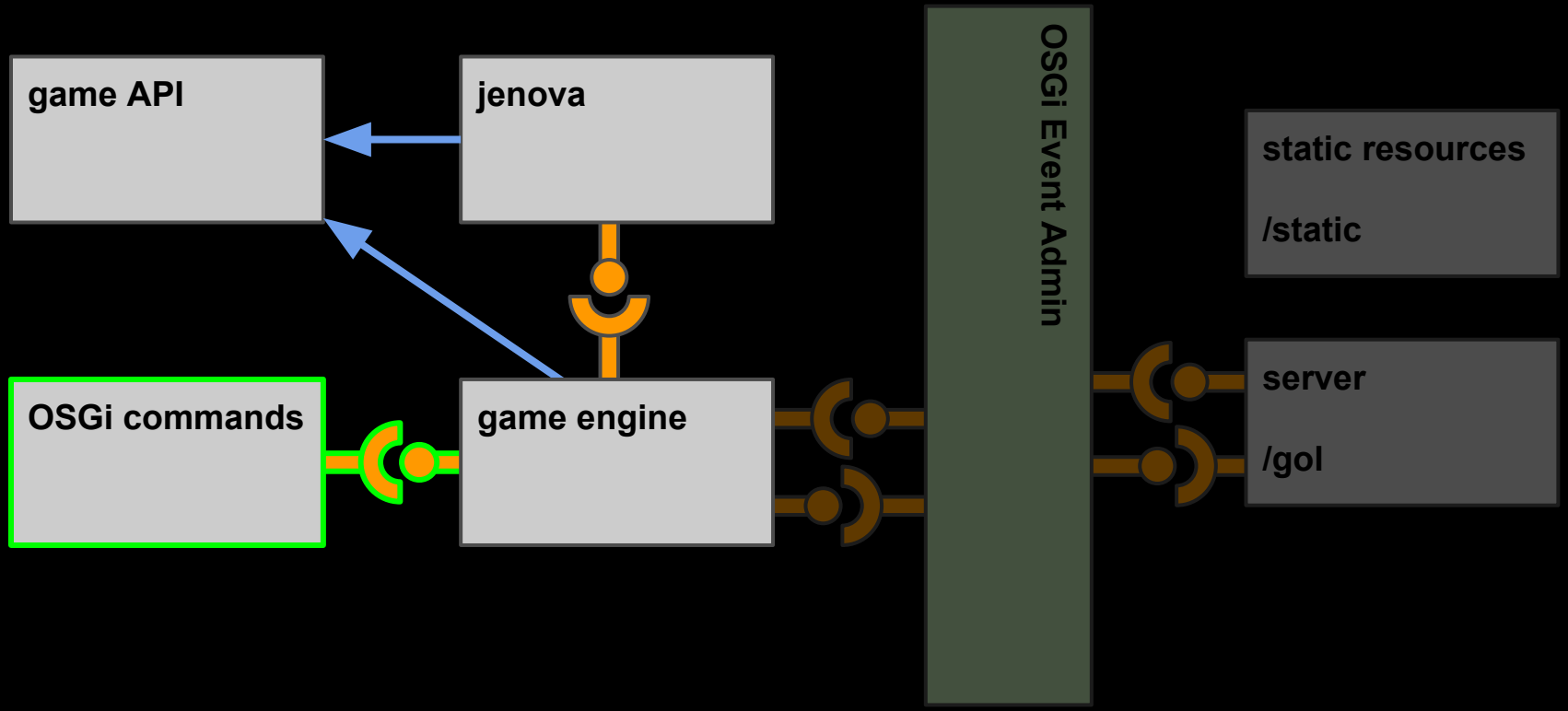
```
git diff task_03_engine_final
```

Bonus NanoService GameEngine

?

Solve the “task” without Annotations - only XML

Task 4: OSGi Game Commands



Start Custom OSGi Commands

```
git checkout task_04_commands_begin
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

Custom OSGi Commands

Provide “**add**” as OSGi commands

```
public class OsgiCommandProvider implements CommandProvider {

  public Object _add(CommandInterpreter commandInterpreter) {
    …
    gameEngine.addObject(...);
    return null;
  }

  public String getHelp() {
    return "...";
  }
}
```

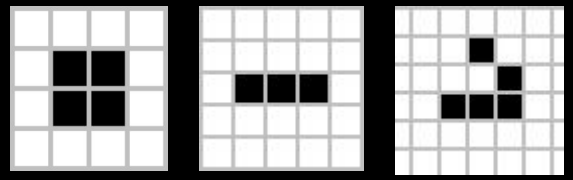
All methods starting with an underscore like “_add” will be available as OSGi commands.

Task 4: OSGi Game Commands

04.1 reference OSGi service `GameEngine`

04.2 implement OSGi command:

```
add [object_name] [x[,y]]
```

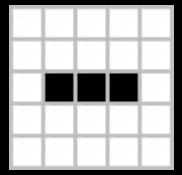


Hint: Predefined patterns are in `OsgiCommandProvider`

Verify Talk to your App on the Shell

```
osgi> init 10 5
osgi> print
    0 1 0 0 0 0 0 0 0 1
    0 1 0 0 0 0 1 0 0 0
    0 0 0 0 0 0 1 0 1 1
    0 0 0 0 0 0 1 0 0 1
    0 1 0 0 0 0 1 1 1 1
```

```
osgi> reset
osgi> add blinker 5 1
osgi> print
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 1 0 0 0 0
    0 0 0 0 0 1 0 0 0 0
    0 0 0 0 0 1 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
```



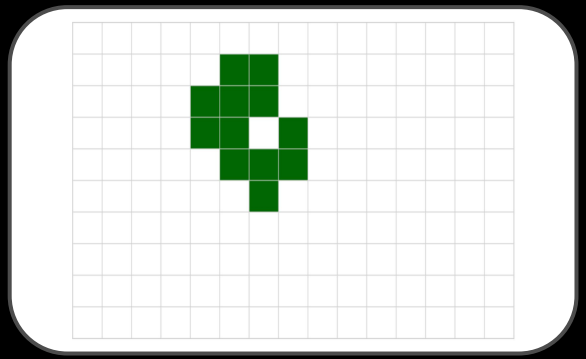
End Custom OSGi Commands

```
git diff task_04_commands_final
```

Bonus Custom OSGi Commands

```
osgi> init 10 1
osgi> print
    0100000111
osgi> flip
osgi> print
    1110000010
```

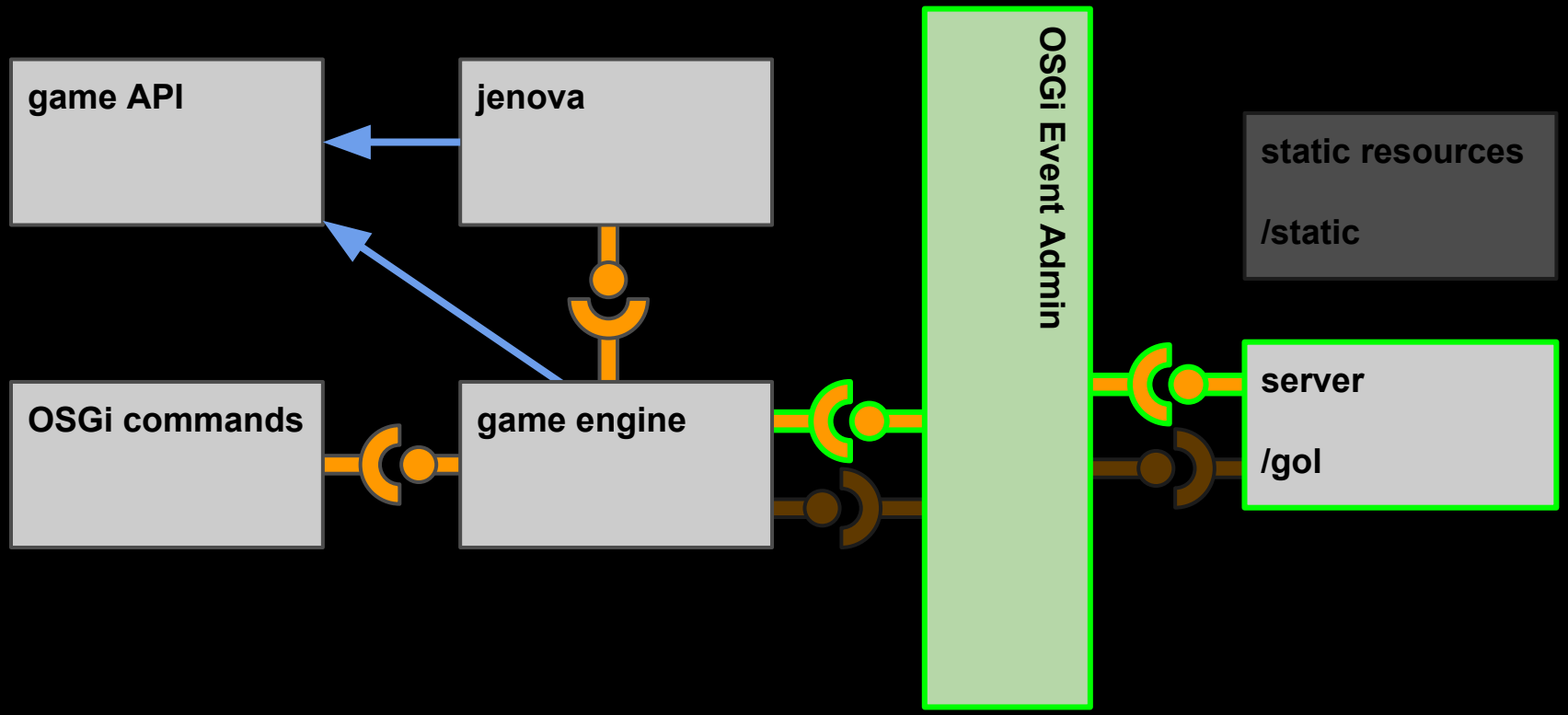
?



Add LWSS (Light Weight Space Ship)

Add command to flip the board vertically

Task 5: Publish Events



Start OSGi Event Admin

```
git checkout task_05_events_begin
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

OSGi EventAdmin + EventHandler

Provides the OSGi EventAdmin as Spring bean

```
<osgi:reference id="eventAdmin" interface="org.osgi.service.event.EventAdmin" />
```

Name of the Spring bean processing the event

```
<osgi:service ref="moveListener" interface="org.osgi.service.event.EventHandler">
  <service-properties>
    <entry key="event.topics" value="topic_foo" />
  </service-properties>
</osgi:service>
```

Only events with this topic will be delivered to the Spring bean

Expose a referenced Spring bean as OSGi **EventHandler** listening for topic **"topic_foo"** with the `<service-properties />` element in XML

Task 5: Publish / Subscribe Events

05.1 autowire `EventAdmin`

05.2 post event "topic_newBoard" with key="board" and payload board

05.3 post event "topic_userModifiedCell" and keys "x", "y"

05.4 register bean moveListener as OSGi service EventHandler for "topic_newBoard" events...

05.5 … and "topic_userModifiedCell" events

05.6 Print events to `System.out` in `MoveListenerDelegate.handleEvent()`

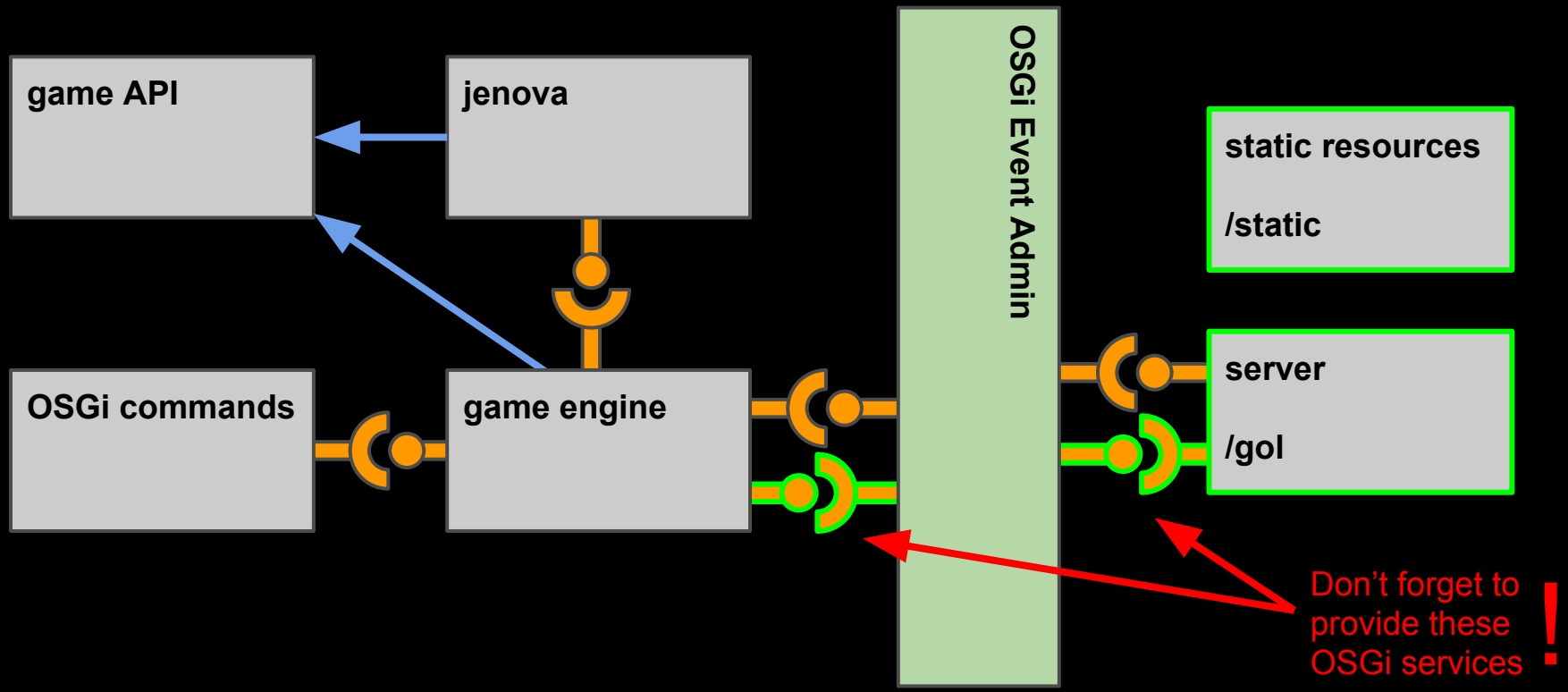
Verify EventAdmin

```
$ tail -f ${VIRGO_HOME}/serviceability/logs/log.log
 -- Calculating next generation --
 Event arrived: o.o.s.e.Event [topic=topic_newBoard]
 Available properties: [board, event.topics]
 -- Calculating next generation --
 Event arrived: o.o.s.e.Event [topic=topic_newBoard]
 Available properties: [board, event.topics]
 ...
```

End OSGi Event Admin

```
git diff task_05_events_final
```

Task 6: WebSocket



Start WebSocket

```
git checkout task_06_websocket_begin
```

Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

<websocket:message-broker />

```
<websocket:message-broker application-destination-prefix="/app">
  <websocket:stomp-endpoint path="/ws">
    <websocket:sockjs />
  </websocket:stomp-endpoint>
  <websocket:simple-broker prefix="/topic" />
</websocket:message-broker>
```

Creates bean **`SimpMessagingTemplate`**

Stomp - text orientated messaging protocol (http://stomp.github.io/)

SockJS - mimics the WebSockets API, but instead of WebSocket there is a SockJS Javascript object. (http://sockjs.org)

Client to Server

```
@Controller("app")
public class App {

  @MessageMapping("/updateCell")
  public void updateCell(Cell cell) {
    // handle incoming message
  }

  @RequestMapping(value = "/board", method = RequestMethod.GET)
  public String board() {
    return "board";
  }
}
```

Called when a user “toggles” a cells

Initial request from the browser

Server to Client

```
@Component("topic")
public class Topic {

  @Autowired
  private SimpMessagingTemplate template;

  public void next(int[][] board) {
    template.convertAndSend("/topic/newBoard", board);
  }
}
```

Provide by `</websocket:message-broker>`

Message payload

WebSocket message destination

Task 6: Websockets

06.1 add message mapping for "/updateCell"

06.2 post event "topic_updateCell" with "x" and "y" coordinates; server side event handling missing

06.3 auto wire `SimpMessagingTemplate`

06.4 convert and send board to "topic/newBoard"

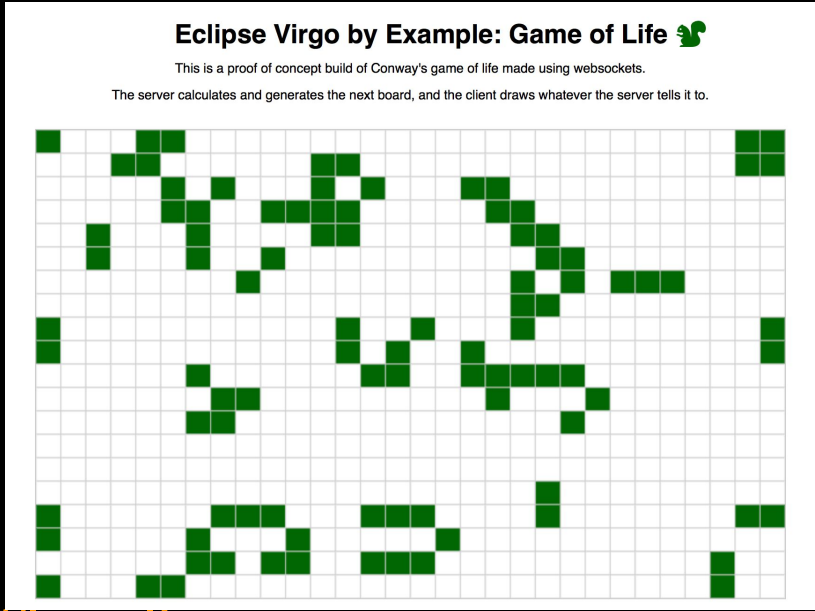
06.5 convert and send cell to "topic/userModifiedCell"

06.6 register stomp endpoint with SockJS support

! **06.7** replace NOP implementation in `MoveListenerDelegate.handleEvent()`

! **06.8** implement `EventHandler` for toggling in `DefaultGameEngine` to enable client to server communication

Verify WebSocket

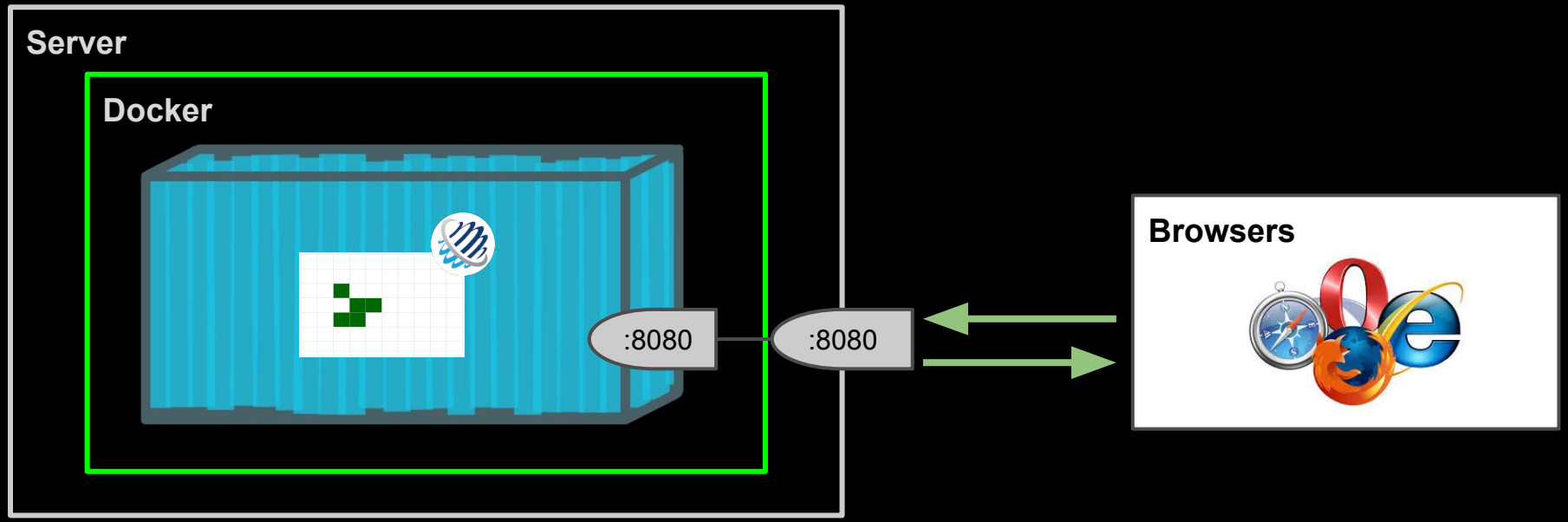


Browse to http://localhost:8080/gol/board

End WebSocket

```
git diff task_06_websocket_final
```

Task 7: Docker Deployment



Start Deployment

```
git checkout task_07_docker_begin
```

USB Stick (Get local copy of base image)

```
$ cat java_openjdk-8u72-jre.tar
| docker load
```

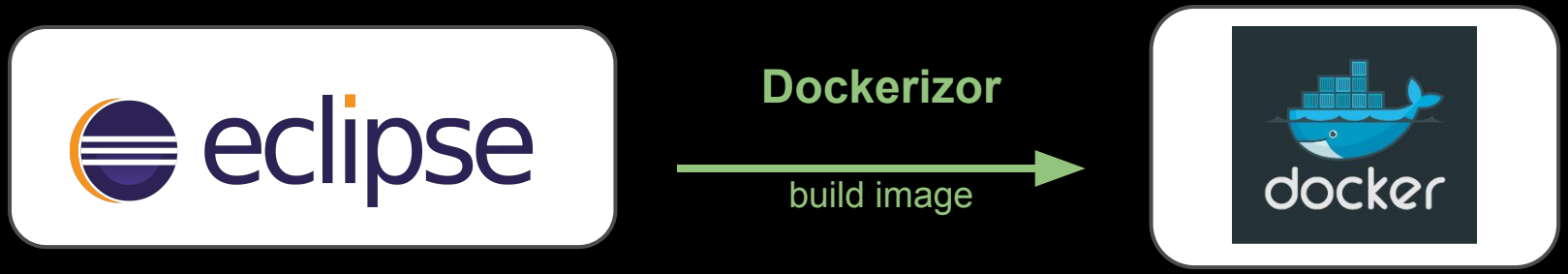
Create Eclipse Project Metadata

```
$ ./gradlew eclipse
```

Import... Gradle Project...

From IDE to Docker Container

- Create Plan file
- Create Dockerizor instructions



<plan />

```
<plan name="game-of-life" version="0.1" scoped="false" atomic="true"...>

  <artifact type="bundle" name="com.eclipsesource.examples.gol.api" version="[0.1, 1)" />
  <artifact type="bundle" name="com.eclipsesource.examples.gol.jenova" version="[0.1, 1)" />
  <artifact type="bundle" name="com.eclipsesource.examples.gol.engine" version="[0.1, 1)" />
  <artifact type="bundle" name="com.eclipsesource.examples.gol.commands" version="[0.1, 1)" />
  <artifact type="bundle" name="com.eclipsesource.examples.gol.server" version="[0.1, 1)" />
  <artifact type="bundle" name="com.eclipsesource.examples.gol.client" version="[0.1, 1)" />

</plan>
```

Plans encapsulate the artifacts of a Virgo application as a single unit.

dockerizor

```
./gradlew dockerize
```

```
dockerizor {
  repository = 'eclipsesource/virgo-tomcat-runtime'
  description = 'Virgo Server for Apache Tomcat'
  virgoFlavour = 'VTS'
}
```

Gradle Plugin Dockerizor

developed at GitHub https://github.com/eclipsesource/dockerizor

available from Gradle Plugins https://plugins.gradle.org/plugin/com.eclipsesource.dockerizor

gradle :runtime-only:dockerize

```
dockerizor {
  repository = 'game-of-life/runtime-only'
  javaImage = 'java:openjdk-8u72-jre'
  hudsonJobName = '3.7.0.M02'
  createLocalCopy = true

  removeAdminConsole = false

  postDockerizeHook = { task ->
     project.logger.info "Adding nashorn packages to configuration/java-server.profile"
     task.RUN "sed -i 's/org.xml.sax.helpers/org.xml.sax.helpers,\\\\\\n jdk.nashorn.api.scripting/' ${project.dockerizor.
virgoHome}/configuration/java-server.profile"
     task.RUN "sed -i 's/ sun.*/ sun.*,\\\\\\n jdk.*/' ${project.dockerizor.virgoHome}/configuration/java-server.profile"
  }
}
```

Name of the generated image

Base image

Creates local copy of the Virgo runtime

gradle :runtime-only:dockerize

Adds 3rd party dependencies to ${VIRGO_HOME}/endorsed/libs

```
dependencies {
  endorsed files('libs/nashorn.jar')

  repositoryExt 'com.fasterxml.jackson.core:jackson-core:2.6.4'
  repositoryExt 'com.fasterxml.jackson.core:jackson-annotations:2.6.3'
  repositoryExt 'com.fasterxml.jackson.core:jackson-databind:2.6.4'
}
```

Adds 3rd party dependencies to ${VIRGO_HOME}/repository/ext

gradle :app:dockerize

Adds the plan to `${VIRGO_HOME}/pickup`

```
dockerizor {
…
  pickupFiles = ['game-of-life.plan']
  dryRun = true
}

dependencies {
…
  repositoryUsr project(':game-api')
  repositoryUsr project(':jenova')
  ...
}
```

No docker daemon on tcp://localhost:4243? Use dry run option to only generate the Dockerfile.

Adds project dependencies to `${VIRGO_HOME}/repository/usr`

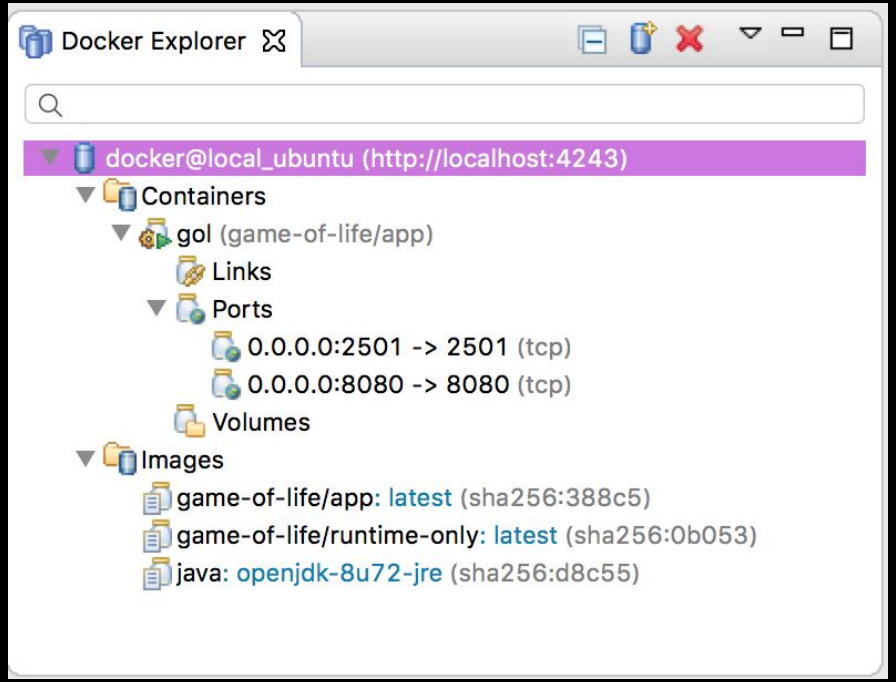
Task 7: Docker Deployment

07.1 search for official Java 8 image at https://hub.docker.com

07.2 add all game-of-life bundles to repositoryUsr

07.3 add game-of-life bundles to Virgo plan file

Verify Deployment



End Deployment

```
git diff task_07_docker_final
```

Bonus Deployment

```
osgi> plan list
osgi>

Name           Version   State
game-of-life   0.1.0     ACTIVE
...
```

Enable OSGi console

Create and run a local copy of “Game-of-Life”

Congratulations, you made it!

Thank you!

eclipsecon
RESTON VIRGINIA 2016

Evaluate the Sessions

Sign in and vote at **eclipsecon.org**
or use our **EclipseCon App**

-1 0 +1

Standard Shell Commands

`lb` list bundles, use `-s` to see symbolic names

`inspect capability service <bundle id>`
show all services provided by a bundle

`start/stop`
start and stop bundles

`grep` same as Unix command (use with pipe `|`)

`headers` print bundle headers

Virgo Shell Commands

`clhas` Lists all bundles that **contain** a class or resource.

`clload` Lists all bundles that can **load** a class.

`plan list`

Lists all plans.

Virgo User Guide: https://www.eclipse.org/virgo/documentation/

Game of Life - Custom Commands

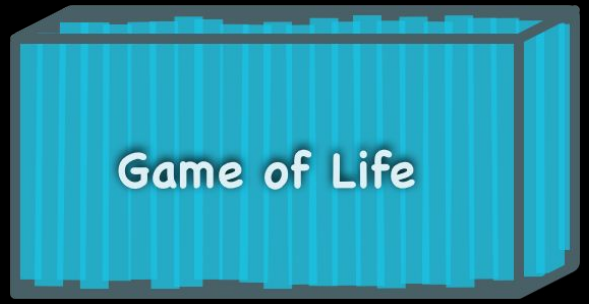
`init [x[,y]]`	initialize a game
`print`	print the current board
`run [ms]`	run the game at the given speed
`pause`	pause the game
`next`	calculate the next generation
`toggle`	toggle state (alive or dead) of a cell

Gradle Build Commands

```
./gradlew <task1> <task2>
```

`build`	build project
`test`	run the tests
`run`	deploy the OSGi bundles via JMX
`dockerize`	create Docker image

`-x <task>` skip a task



Docker Commands

`docker`

`build`	Build a new image
`run`	Create a new container and start it

[build] https://docs.docker.com/engine/reference/commandline/build/
[run] https://docs.docker.com/engine/reference/run/