Integration Tests using Docker, Scala, Akka & ScalaTest

WoQ
WAY OF QUALITY

Andreas Gies
Lead Architect

- Technical Consulting
- Conference speaking slots

- Technical Trainings
- Management Seminars
- Team Breakouts

Coding by the sea

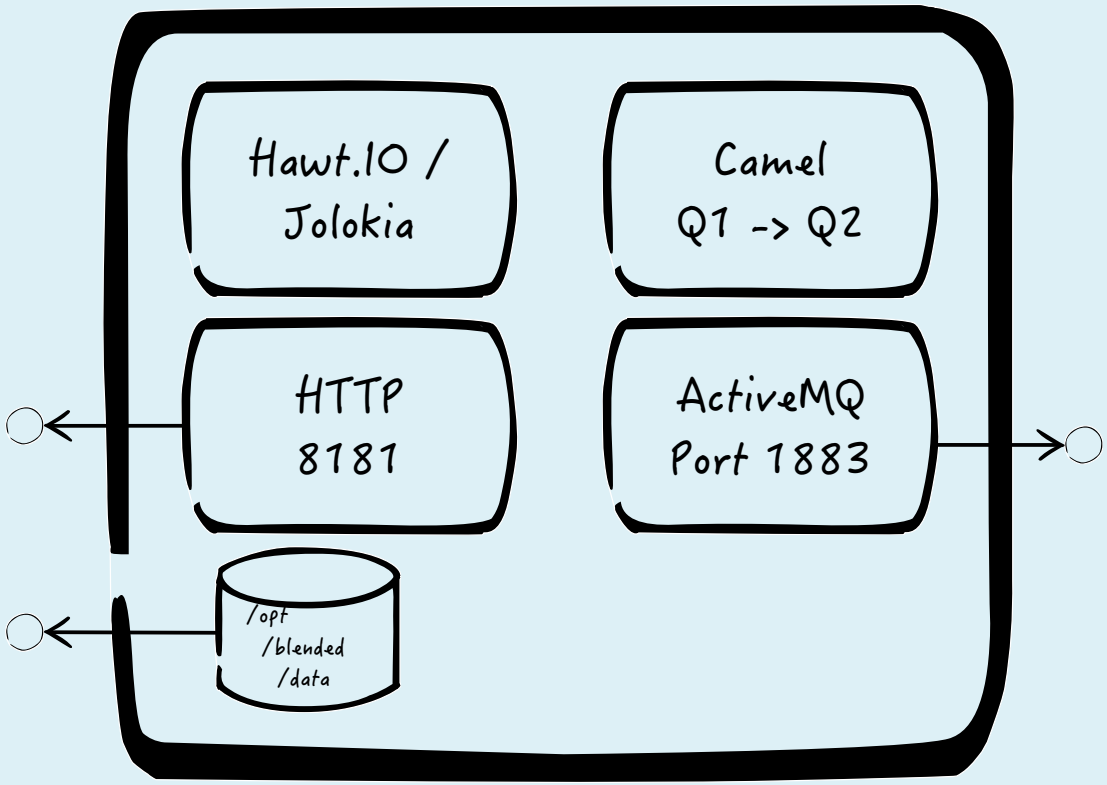
Yoga & learning

Team Building

Enjoy Andalucia

SAMPLE TEST

A sample container under test



An example test

```
"The demo container" should {

  "Define the sample Camel Route from SampleIn to SampleOut" in {

    implicit val camelContext = testContext

    withTestContext { ctxt =>
      ctxt.withMock("sampleOut", "jms:queue:SampleOut")
      ctxt.start()

      val mock = ctxt.mockEndpoint("sampleOut")
      mock.setExpectedMessageCount(1)
      mock.expectedBodyReceived().constant("Hello Blended!")

      ctxt.sendTestMessage("Hello Blended!", "jms:queue:SampleIn")
      mock.assertIsSatisfied(2000l)

      None
    }
  }
}
```

The example test context

Provide a Camel based test context with all required components for testing

```
private def testContext = {
  val result = new TestCamelContext()
  result.withComponent(
    "jms", JmsComponent.jmsComponent(
      BlendedTestContext(
        BlendedDemoIntegrationSpec.amqConnectionFactory)
          .asInstanceOf[ConnectionFactory]
    )
  )
  result
```

The container under test

```
docker {
  url:       "http://${docker.host}:${docker.port}"
  user:      "atooni"
  password:  "foo"
  eMail:     "andreas@wayofquality.de"
  version:   "1.12"

  containers : [
    {
      name:     "blended_demo"
      image:    "^.*/blended_demo:latest"
      volumes: [
        {
          host : "data"
          container : "/opt/blended/data"
        }
      ]
      ports : [
        { name: "jms",  value: "1883" },
        { name: "http", value: "8181" },
        { name: "jmx",  value: "1099" }
      ]
    }
  ]
}
```

- Docker Connector
- Container Reference
- Exported Volumes
- Exported ports

Providing the JMS connection

- We know the JMS port is 1883 and Active MQ is the JMS provider
- Keep information relevant across Specs in an instance of BlendedTestContext

```
private lazy val amqConnectionFactory = {
  val ctInfo  = Await.result(containerInfo("blended_demo_0"), 3.seconds)
  val address = ctInfo.getNetworkSettings.getIpAddress

  val brokerUrl = s"tcp://${address}:1883"

  system.log.info(s"Using AMQ connection url [${brokerUrl}]")

  BlendedTestContext.set(
    BlendedDemoIntegrationSpec.amqConnectionFactory,
    new ActiveMQConnectionFactory(brokerUrl)
  ).asInstanceOf[ConnectionFactory]
}
```

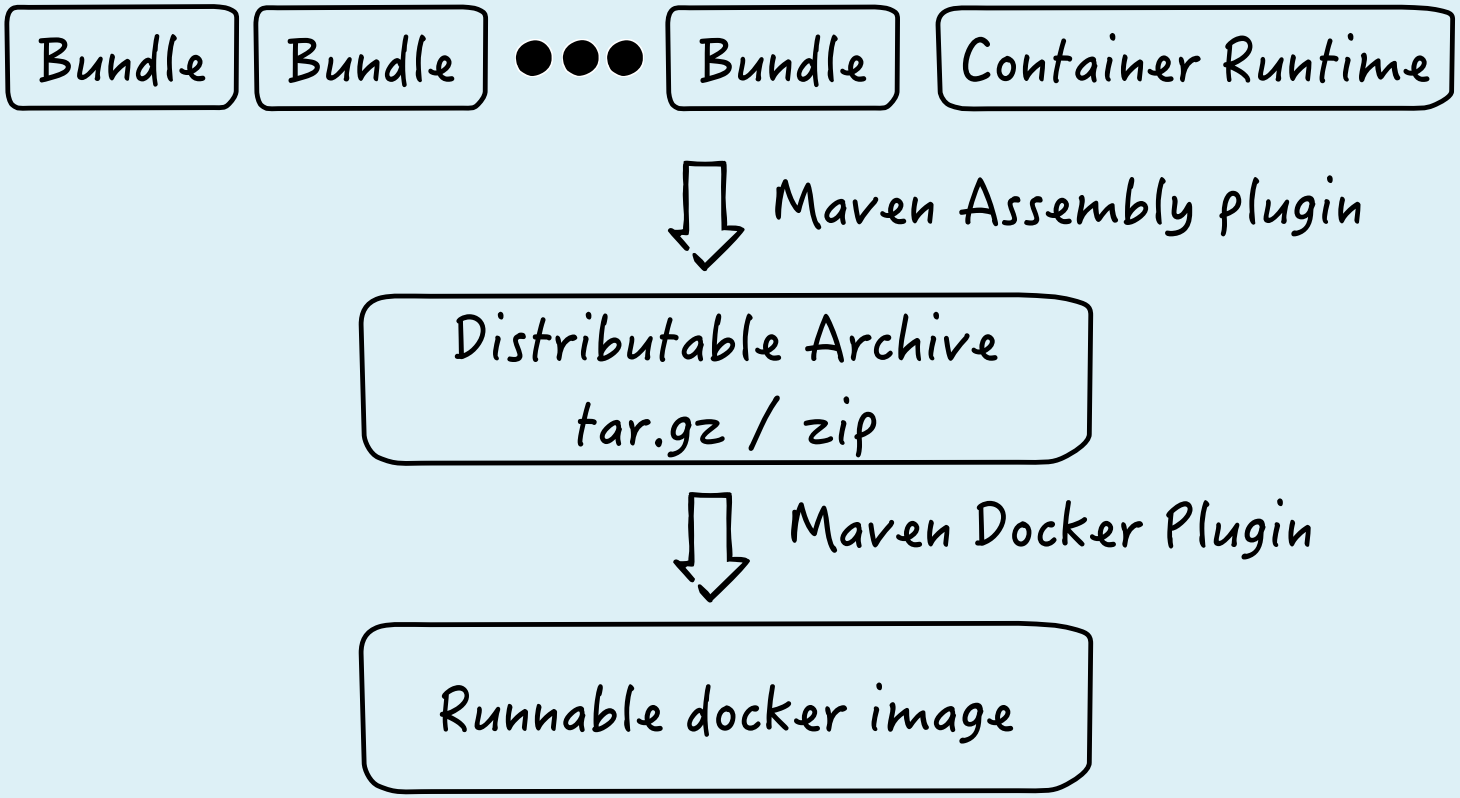
Waiting for the container

Wait until Jolokia, JMS and the Camel Route under test are available

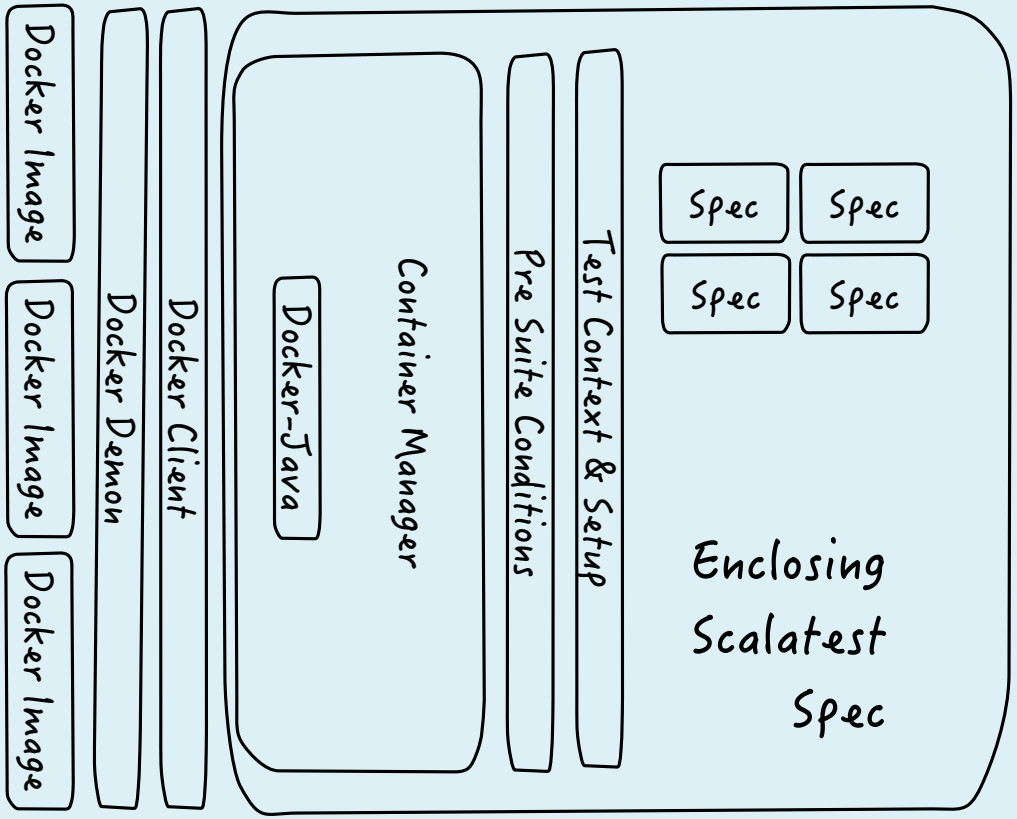
```
override def preCondition = {
  val t = 60.seconds

  SequentialComposedCondition(
    ParallelComposedCondition(
      JolokiaAvailableCondition(jmxRest, Some(t), Some("blended"), Some("blended")),
      JMSAvailableCondition(amqConnectionFactory, Some(t))
    ),
    CamelContextExistsCondition(
      jmxRest, Some("blended"), Some("blended"), "BlendedSample", Some(t)
    )
  )
}
```

WRITING TESTS

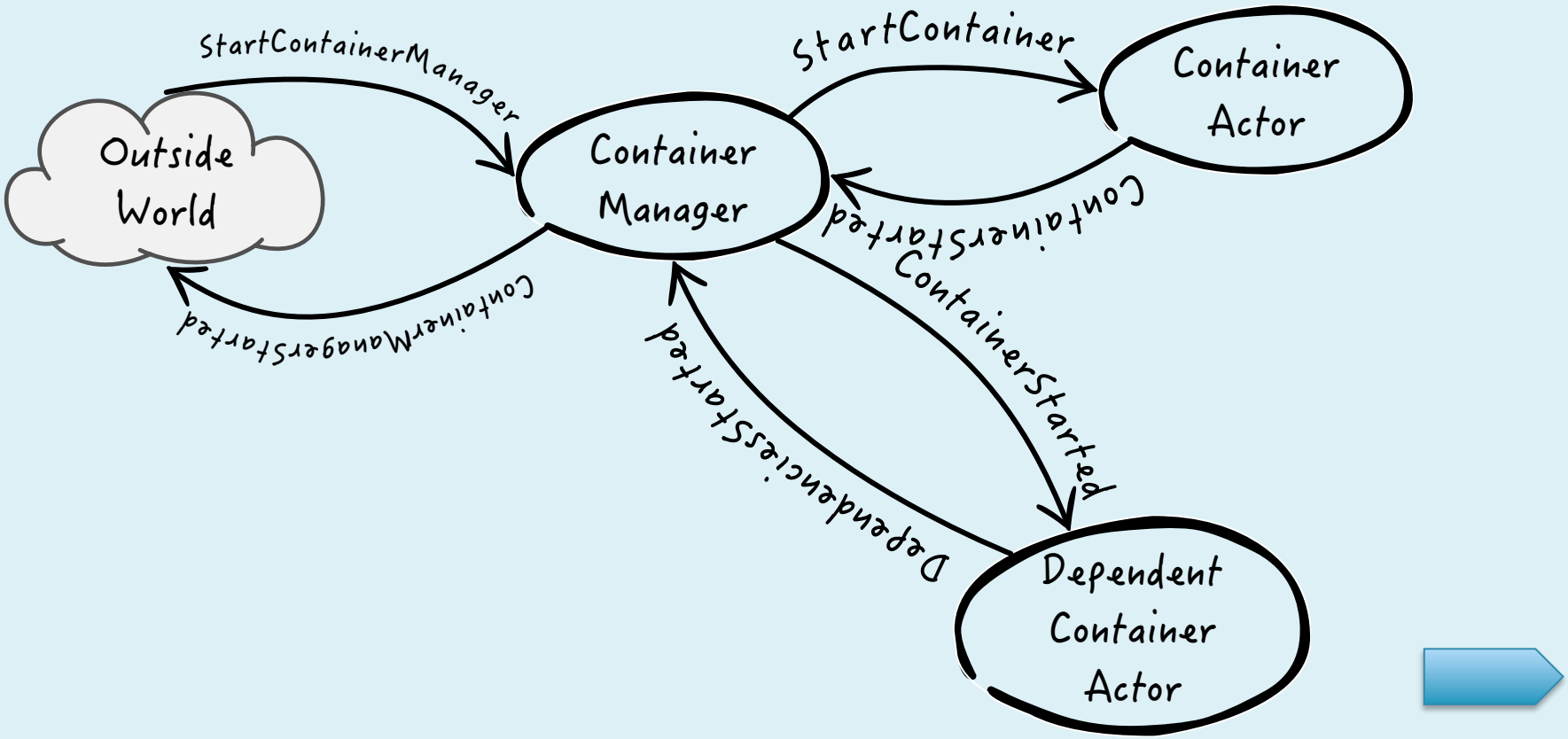
Test packaging



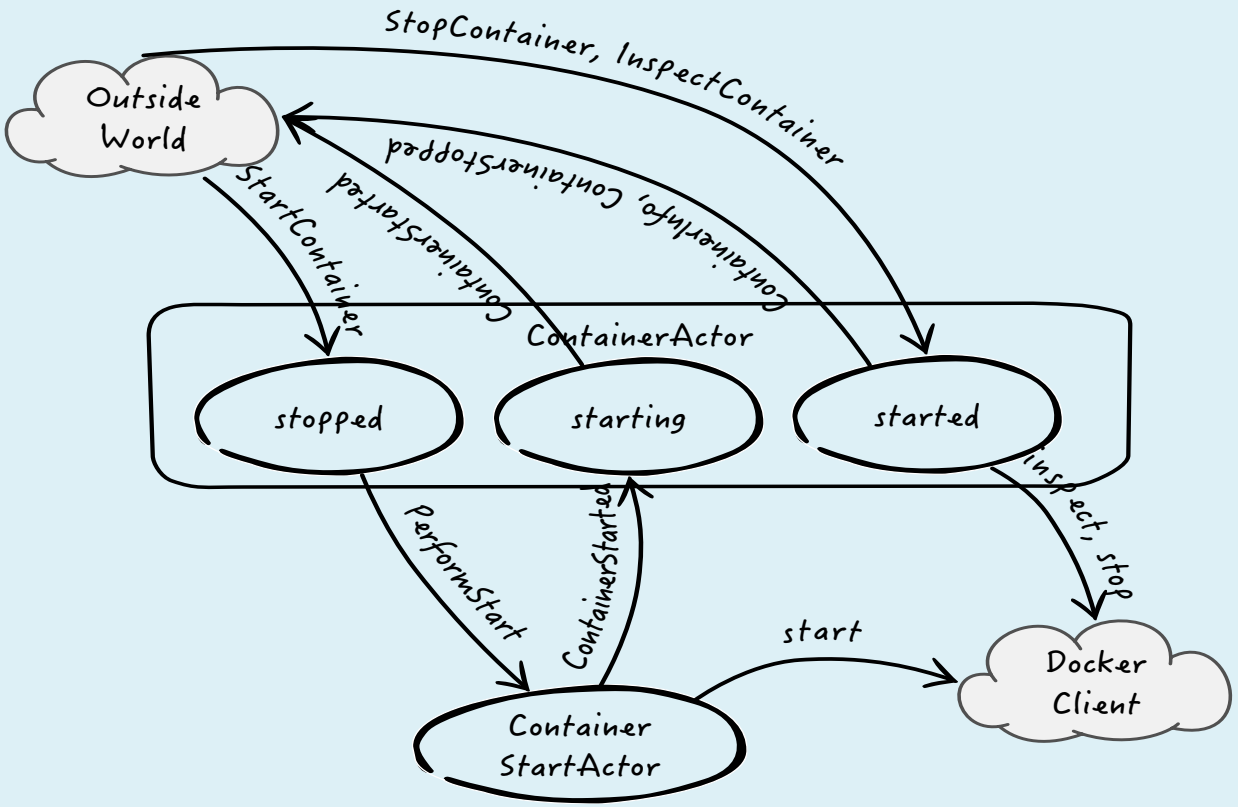
High Level Test Architecture



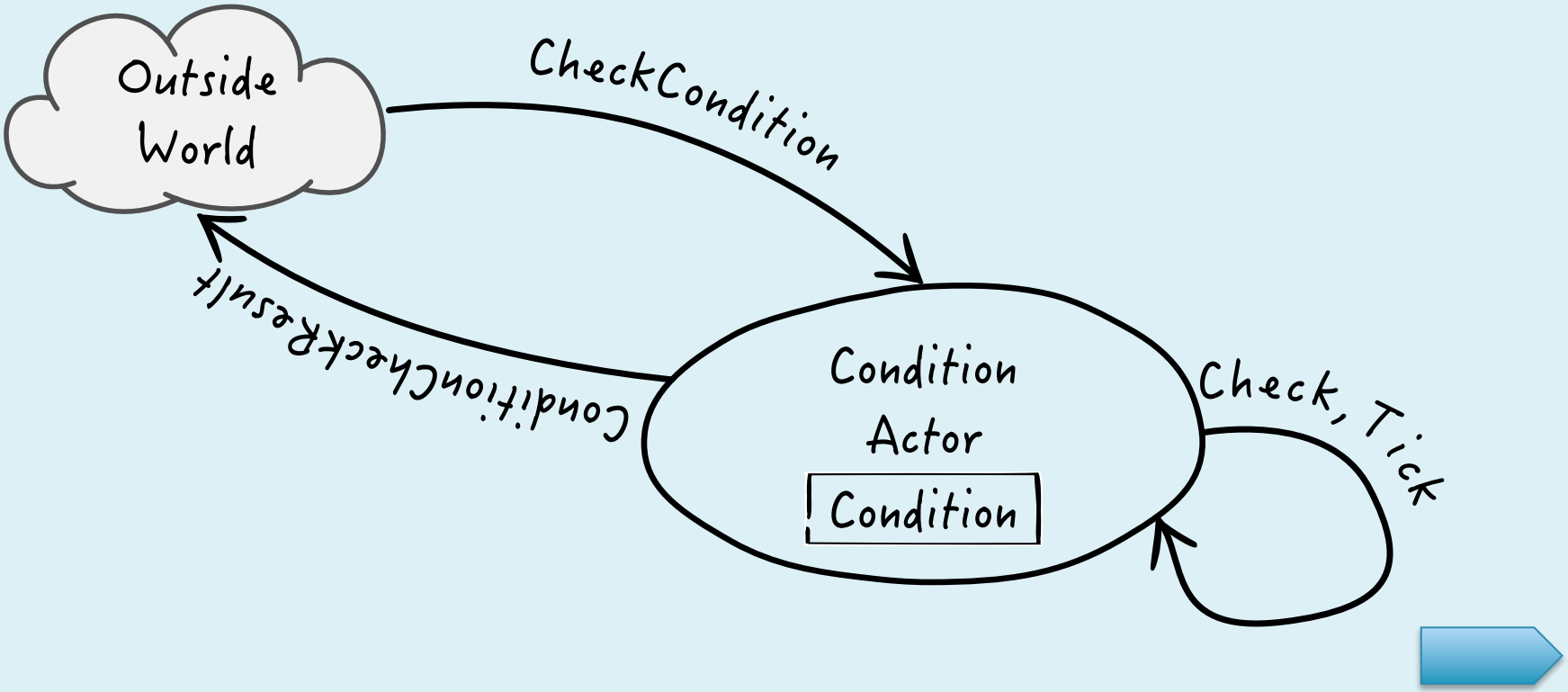
An Akka based Container Mgr



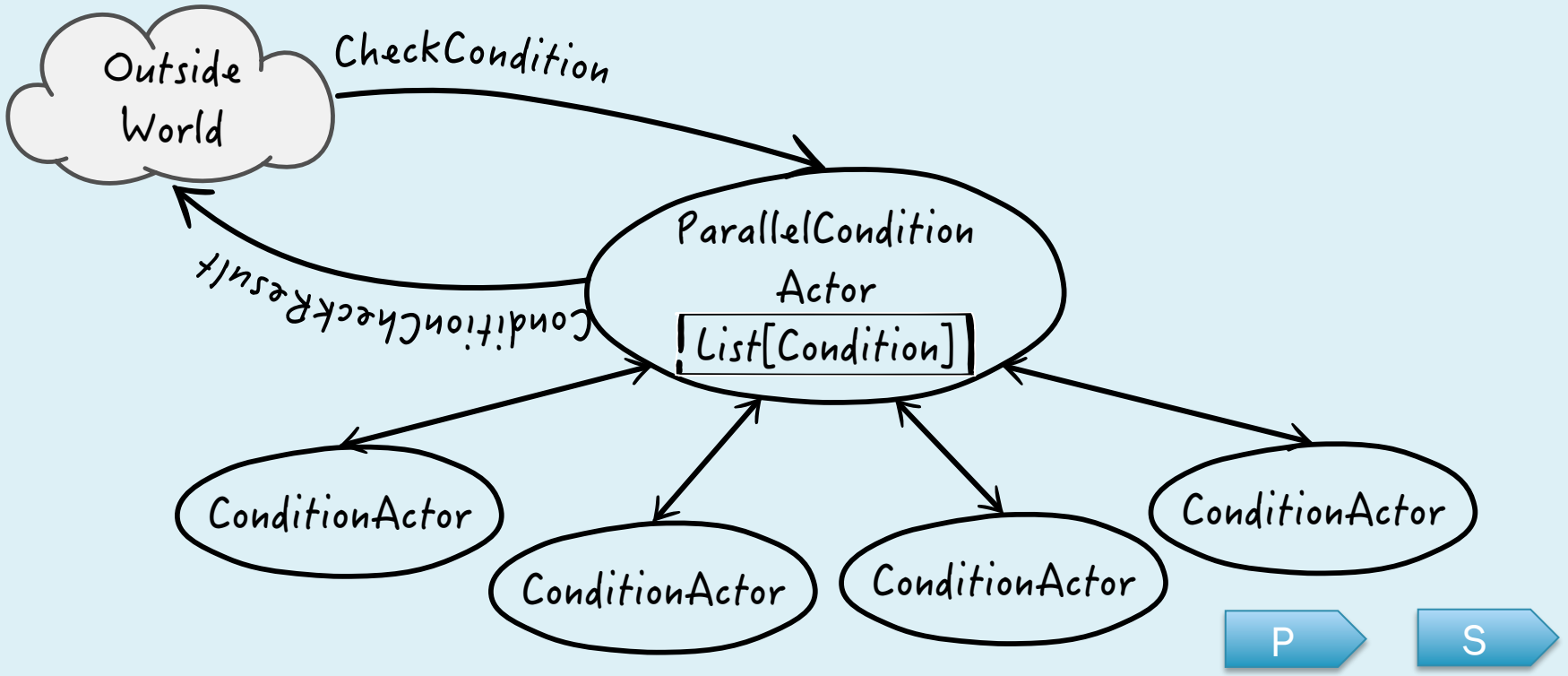
Container Actor



Managing Conditions



Composed Conditions



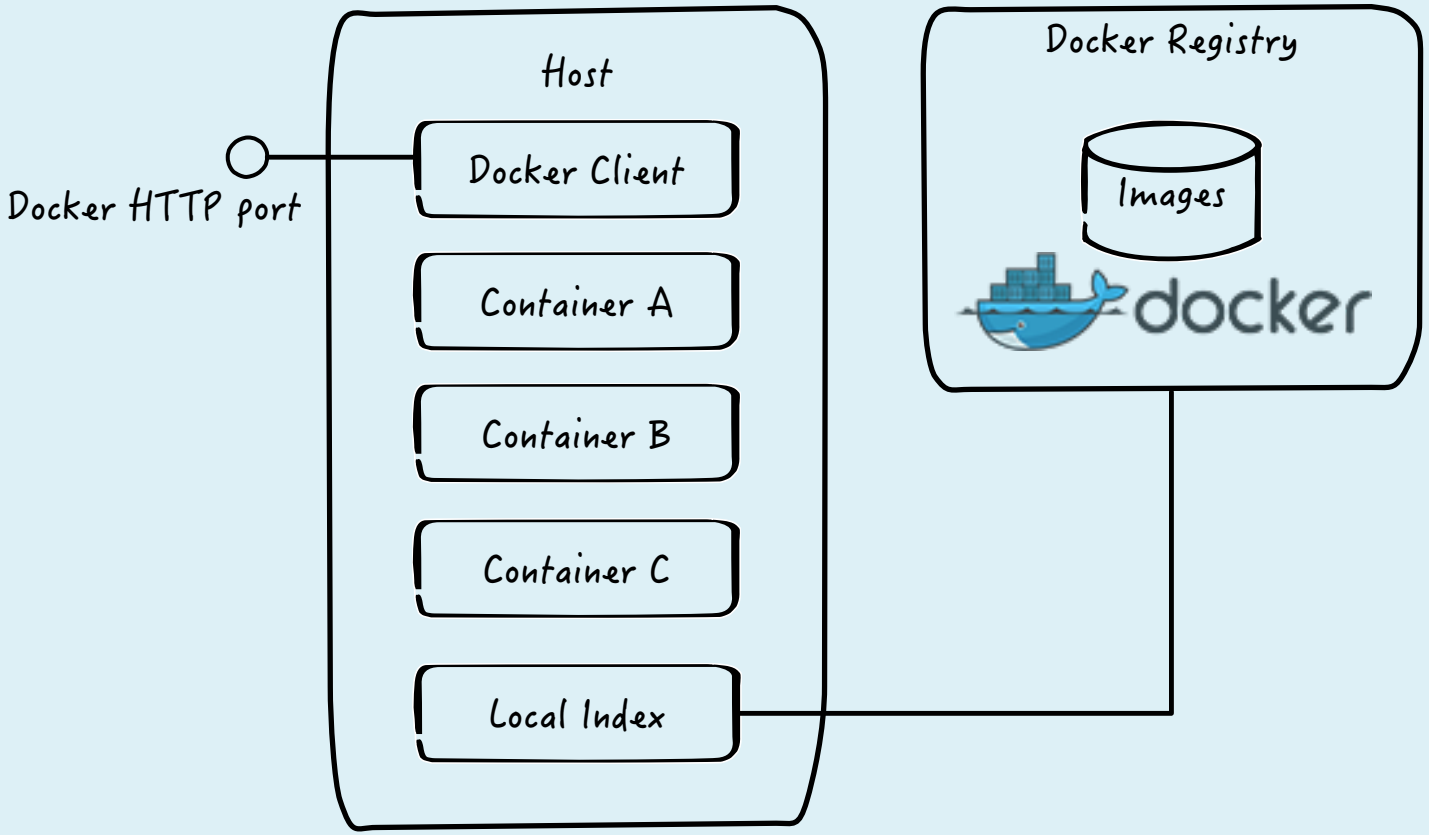
Composing Conditions

```
object ConditionActor {
  def apply(cond: Condition) = cond match {
    case pc : ParallelComposedCondition => new ParallelConditionActor(pc)
    case sc : SequentialComposedCondition => new SequentialConditionActor(sc)
    case _ => new ConditionActor(cond)
  }
}
```

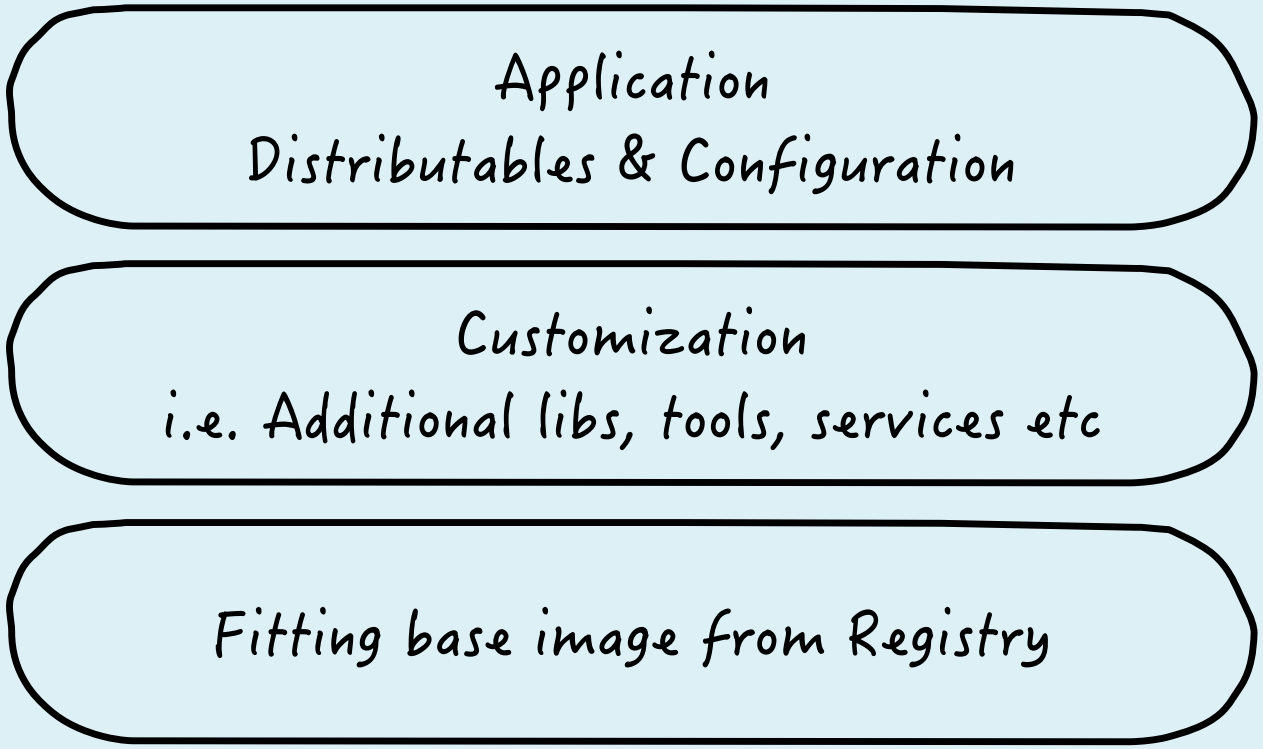
```
abstract class ComposedCondition(condition: Condition*) extends Condition {

  private var isSatisfied : AtomicBoolean = new AtomicBoolean(false)
  override def satisfied = isSatisfied.get()
}

case class SequentialComposedCondition(conditions: Condition*) extends ComposedCondition(conditions.toSeq:_*) {
  override def timeout = conditions.foldLeft(interval * 2)( (sum, c) => sum + c.timeout)
  override val description = s"SequentialComposedCondition(${conditions.toList})"
}

case class ParallelComposedCondition(conditions: Condition*) extends ComposedCondition(conditions.toSeq:_*) {
  override def timeout = (conditions.foldLeft(interval * 2)((m, c) => if (c.timeout > m) c.timeout else m)) + interval * 2
  override val description = s"ParallelComposedCondition(${conditions.toList})"
}
```

CREATING DOCKER IMAGES

Docker in a nutshell



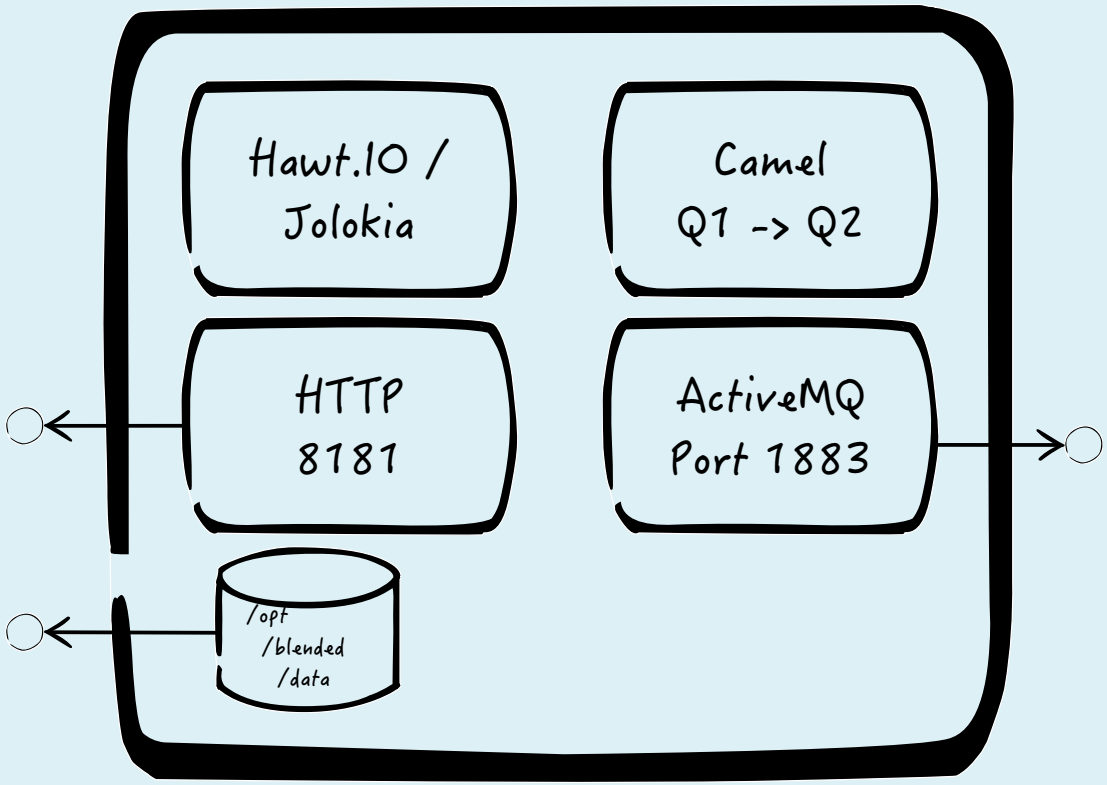
Layering Docker images



Basic Container Test image

```
# The basic image to run Blended containers
FROM centos:6.4

MAINTAINER Andreas Gies version: 1.0.7-M6-SNAPSHOT

# Installation section
RUN groupadd blended
RUN useradd -d /home/blended -s /bin/bash -g blended -m
blended

ADD files/jdk-7u60-linux-x64.tar.gz /opt
RUN ln -s /opt/jdk1.7.0_60 /opt/java

# End of Installation section
```

Deriving the image under test



Container configuration

```
FROM atooni/blended-base:latest
ADD blended-karaf-demo-${project.version}-nojre.tar.gz /opt
RUN ln -s /opt/blended-karaf-demo-${project.version} /opt/blended
RUN mkdir -p /opt/blended/data
RUN chown -R blended.blended /opt/blended*
USER blended
ENV JAVA_HOME /opt/java
ENV PATH ${PATH}:${JAVA_HOME}/bin
ENTRYPOINT ["/opt/blended/bin/karaf"]
VOLUME /opt/blended/data
EXPOSE 1883
EXPOSE 8181
EXPOSE 1099
```

Pom File definitions

- Dependency on distributable archive
- Reference Docker definition file
- Use dependency plugin to retrieve archive
- Invoke docker plugin

Docker tricks

- Configuring docker for integration tests
  - Creating the TCP-IP Connector (/etc/sysconfig/docker)
- Cleaning up docker

WRITING TESTS

General tasks to write tests

- Define the containers under test
- Waiting for Containers to be started
- Connecting to the Containers
- Test fixtures
- Boilerplate test context

A slightly bigger use case

```
{
    name:   "gs4"
    image:  "^kaufland_sibfile:latest"
    links:  [ "shop_0:shop" ]
    ports: [
      { name: "http", value: "8181" }
    ]
    volumes: [
      {
        host: "data"
        container: "/opt/sibfile/data"
      },
      {
        host: "tmp"
        container: "/tmp"
      }
    ]
  }
```

```
op"
aufland_shop:latest"
ic_0:sonic" ]

ta"
r : "/opt/shop/data"

p",   value: "4747" },
ttp",  value: "8777" },
ms",   value: "1883" }
```

```
e:   "sonic"
e:  "^kaufland_sonic:latest"
s : [
me: "data", value: "2620" },
me: "mgmt", value: "2608" }
```

Wait until the CuT are ready

```
override def preCondition: Condition = SequentialComposedCondition(
  ParallelComposedCondition(
    JolokiaAvailableCondition(
      shopJolokia, Some(120.seconds), Some("kaufland"), Some("kaufland")
    ),
    JolokiaAvailableCondition(
      gs4Jolokia, Some(120.seconds), Some("kaufland"), Some("kaufland")
    ),
    JMSAvailableCondition(
      activeMQConnectionFactory, Some(120.seconds)
    ),
    JMSAvailableCondition(
      sonicConnectionFactory, Some(300.seconds)
    )
  ),
  CamelContextExistsCondition(
    gs4Jolokia, Some("kaufland"), Some("kaufland"), "fileServices-ro-gs4",
    Some(120.seconds)
  )
)
```

Connect to the CuT's

```
private lazy val shopAddress =
  Await.result(containerInfo(shopContainerName), 3.seconds).getNetworkSettings.getIpAddress

private lazy val sonicAddress =
  Await.result(containerInfo(sonicContainerName), 3.seconds).getNetworkSettings.getIpAddress

private lazy val shopJolokia = {
  val url = Await.result(jolokiaUrl(ctName = shopContainerName, port = 8777), 3.seconds)
  url should not be (None)
  BlendedTestContext.set(ShopIntegrationSpec.shopJolokia, url.get).asInstanceOf[String]
}

private lazy val gs4Jolokia = {
  val url = Await.result(jolokiaUrl(ctName = gs4ContainerName, port = 8181), 3.seconds)
  url should not be (None)
  BlendedTestContext.set(ShopIntegrationSpec.gs4Jolokia, url.get).asInstanceOf[String]
}
```

Connect to the CuT's

```
private lazy val activeMQConnectionFactory = {
  BlendedTestContext.set(
    ShopIntegrationSpec.amqConnectionFactory,
    new ActiveMQConnectionFactory(s"tcp://${shopAddress}:1883")
  ).asInstanceOf[ConnectionFactory]
}

private lazy val sonicConnectionFactory = {
  BlendedTestContext.set(
    ShopIntegrationSpec.sonicConnectionFactory,
    new progress.message.jclient.ConnectionFactory(
      s"tcp://${sonicAddress}:2620", "Administrator", "Administrator"
    )
  ).asInstanceOf[ConnectionFactory]
}

private lazy val buildFTPUri = {

  import de.woq.kaufland.sib.itest.shop.ShopConfig._

  BlendedTestContext.set(
    ShopIntegrationSpec.ftpUrl,
    s"ftp://${FTP_USER}@${shopAddress}:4747?password=${FTP_PWD}&passiveMode=true&binary=true"
  ).asInstanceOf[String]
}
```

Text fixtures

```
private def createQueues(queues : List[String]) {

  implicit val timeout = new Timeout(3.seconds)

  val mfConnector = new SonicDomainConnector(
    countryName = "central",
    domain = "dmModel",
    url = s"tcp://${sonicContainerInfo.getNetworkSettings.getIpAddress}:2608",
    user = "Administrator",
    pwd = "Administrator"
  )

  val mfUtils = system.actorOf(Props(MFUtilsWorker(mfConnector)))

  queues.foreach { queue =>

    val request = (mfUtils ? MFUtilsRequest("central", SonicMgmtTasks.createQueueTask(
      target = "/Clusters/clData",
      cluster = true,
      config = new QueueConfig(name = queue, maximumSize = 10240, saveThreshold = 15360)
    ))).mapTo[MFUtilsResponse[QueueConfig]]


    val response = Await.result(request, 3.seconds)
  }

  mfUtils ! PoisonPill
}
```

A Boilerplate Test Context

```
class TestCamelContext() extends DefaultCamelContext with CamelTestSupport {

  private val log = LoggerFactory.getLogger(classOf[TestCamelContext])

  def mockEndpoint(name: String) = getEndpoint(s"mock://${name}").asInstanceOf[MockEndpoint]

  def withComponent(compName: String, component: Component) = {
    log.debug(s"Adding component [${compName}] to TestCamelContext")
    addComponent(compName, component)
    this
  }

  def withMock(mockName: String, mockUri: String) = {
    log.debug(s"Adding mock endpoint and route [${mockName}] to TestCamelContext.")

    addRoutes( new RouteBuilder() {
      override def configure() {
        from(mockUri).id(mockName).to(s"mock://${mockName}")
      }
    })
    this
  }
}
```

CamelTestSupport

- Create & send test messages
- Wire mock Endpoints
- Frequently used message assertions

Injecting components

```
def testContext = {
  val result = new TestCamelContext()

  result
    .withComponent("sonic", JmsComponent.jmsComponent(sonicConnectionFactory))
    .withComponent("activemq", JmsComponent.jmsComponent(activeMQConnectionFactory))
    .withComponent("ftp", new FtpComponent())

  result
}
```

Finally – The spec

```
"Drop files sent from the data center into the configured target directory" in {

  val gs4DropDir = volumeBaseDir + "/gs4_0/tmp/gs4"
  implicit val camelContext = testContext

  withTestContext { ctxt =>

    ctxt
      .withMock("gs4", s"file://${gs4DropDir}")
      .withMock("event", cbeOutUri(COUNTRY))
      .start()

    val gs4Mock = ctxt.mockEndpoint("gs4")
    gs4Mock.setExpectedMessageCount(1)
    gs4Mock.expectedBodiesReceived("Hello GS4")

    val event = ctxt.mockEndpoint("event")
    event.setExpectedMessageCount(2)

    ctxt.sendTestMessage(
      "Hello GS4",
```

More test support

- Akka based JMS producer/consumer for throughput measurements (Early stage)
- Akka based Jolokia client to Query the container's JMX resources via REST

INTEGRATION TEST GUIDELINES

Integration Test Guidelines

- Concentrate on Blackbox Tests
  - Aka Don't modify the CuT to test it …
- Identify the Communication endpoints
  - Aim for readable Specs and discuss with end users
- Specify Green and Red Path Tests
- Try to capture load scenarios
  - Hard to define due to machine dependencies
- Test fixtures
  - Try to abstract test fixtures into abstract classes

DEMO

CONCLUSION

More to come...

Working

- Container Management
- Test Framework with easy hooks to CuT
- Implicit Test Context for Blackbox testing
- Syntactic sugar and support for Camel based black-box testing

Wishlist / Issues

- More syntactic sugar for Conditions
- Networking issue on OS X
- Kill Containers throughout test for failover testing
- Leverage a hosting env. like Openshift to execute Containers under test
- Publish Test Framework as self-sufficient project

Resources

- GitHub Project Blended
  - Travis Build – Waffle Board – Codacy Code metrics - Project Page
- Way of Quality Homepage
- Scala – Akka – Docker – ScalaTest
- Apache Camel - for writing test routes
- Apache Karaf - The containers under test in the samples
- Apache ActiveMQ - The JMS layer used in the samples
- Hawtio – The management console of the test containers
- Jolokia – A JMX to REST bridge

Stay in contact !

- andreas@wayofquality.de
- http://de.linkedin.com/pub/andreas-gies/0/594/aa5
- https://www.facebook.com/andreas.gies.5
- @andreasgies, #WOQ-Blended on Twitter

- Enjoy a course or a coding breakout

Castillo San Rafael
Andalusian Centre for Art & Cultural Holidays