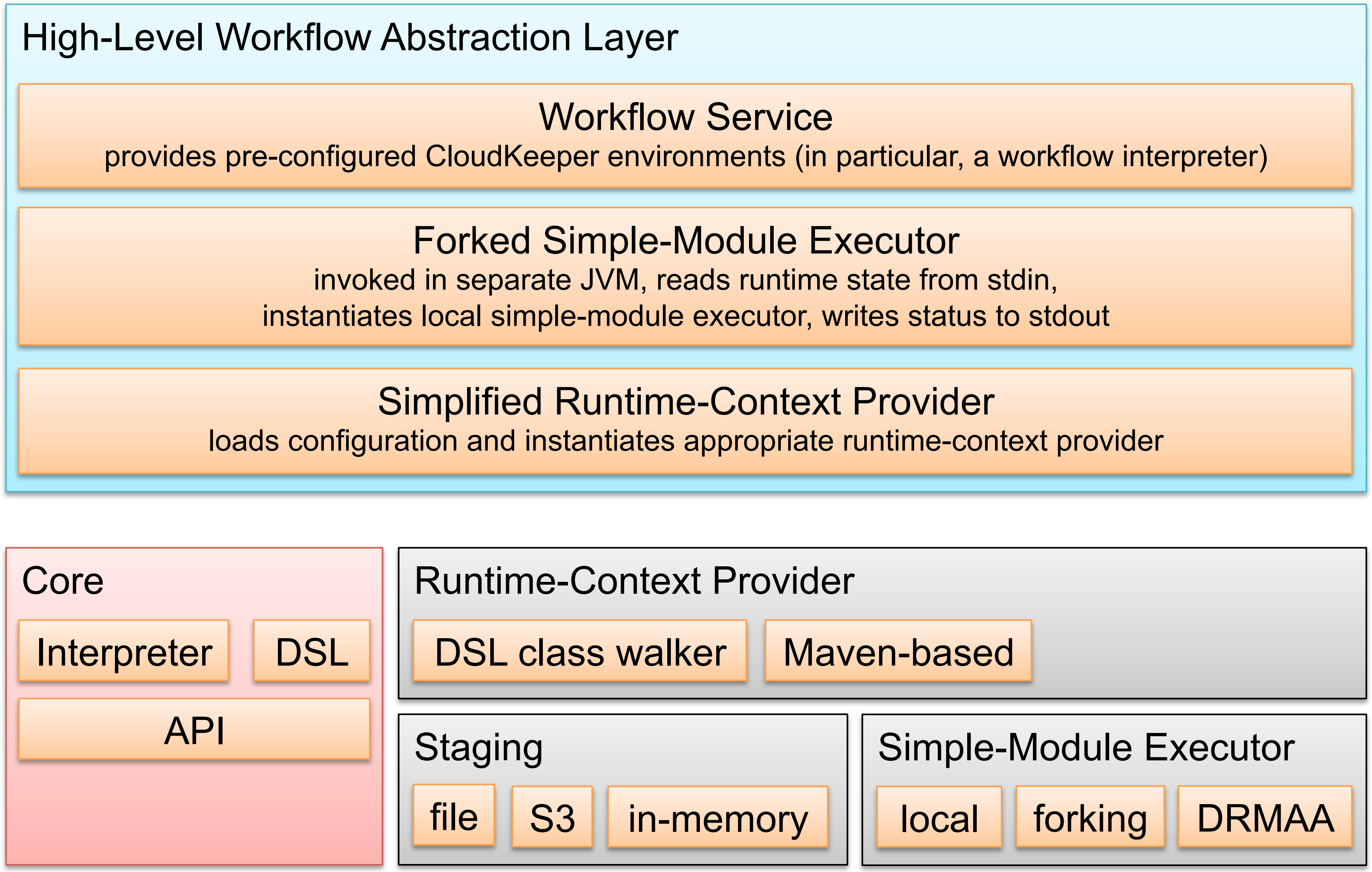
High-Level Wrapper for CloudKeeper

Architecture

Configuration

Architecture


High-Level Workflow Abstraction Layer

- **Workflow Service** — provides pre-configured CloudKeeper environments (in particular, a workflow interpreter)
- **Forked Simple-Module Executor** — invoked in separate JVM, reads runtime state from stdin, instantiates local simple-module executor, writes status to stdout
- **Simplified Runtime-Context Provider** — loads configuration and instantiates appropriate runtime-context provider

lifecode

Configuration of CloudKeeper Abstraction Layer

Typesafe Config library / HOCON

- “Human-Optimized Config Object Notation”
- See GitHub Wiki Page

```
com.svbio.workflow {
    # Settings pertaining to database logging.
    database {
        # The schema (table qualifier) in which the database tables reside.
        schema = "public"

        # Java Persistence API 2.1 properties. All properties in this group
        # will be passed to method Persistence#createEntityManagerFactory
        # as-is (without the "com.svbio.workflow.database" prefix).
        javax.persistence {
            jdbc.driver = org.h2.Driver
            jdbc.url = "jdbc:h2:mem:workflowservice"
            jdbc.user = ""
            jdbc.password = ""
            schema-generation.database.action = create
        }
    }
```

lifecode

Runtime-Context Provider

DSL class walker

`com.svbio.workflow.loader = dsl`

- `x-cloudkeeper-dsl:<`*`class name`*`>`
  - Class name must be DSL module class
  - No dynamic class-loader creation
  - under the hood: creates bundle with all transitively referenced plug-in declarations

Maven-based

`com.svbio.workflow.loader = aether`

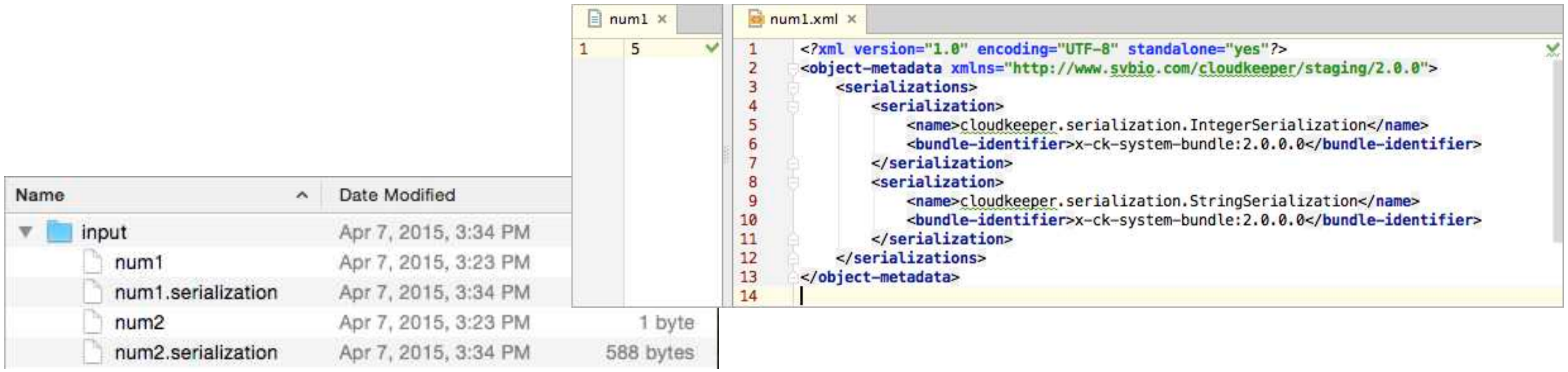
- `x-maven:<`*`groupId`*`>:<`*`artifactId`*`>:ckbundle[:<`*`classifier`*`>]:<`*`version`*`>`
- Dynamic class-loader creation by default
  - can be deactivated by manually instantiating `MavenRuntimeContextFactoryModule`

Staging-Area Provider

File System

`com.svbio.workflow.staging = file`

- new directory for each workflow execution
- directly under configured base path
- file or directory for each in-/out-port
- file “x.meta.xml” for each port *x*



```
Name                      Date Modified
▼ input                   Apr 7, 2015, 3:34 PM
    num1                  Apr 7, 2015, 3:23 PM
    num1.serialization    Apr 7, 2015, 3:34 PM
    num2                  Apr 7, 2015, 3:23 PM        1 byte
    num2.serialization    Apr 7, 2015, 3:34 PM     588 bytes
```

num1

```
5
```

num1.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<object-metadata xmlns="http://www.svbio.com/cloudkeeper/staging/2.0.0">
    <serializations>
        <serialization>
            <name>cloudkeeper.serialization.IntegerSerialization</name>
            <bundle-identifier>x-ck-system-bundle:2.0.0.0</bundle-identifier>
        </serialization>
        <serialization>
            <name>cloudkeeper.serialization.StringSerialization</name>
            <bundle-identifier>x-ck-system-bundle:2.0.0.0</bundle-identifier>
        </serialization>
    </serializations>
</object-metadata>
```

Amazon S3

`com.svbio.workflow.staging = s3`

- like file staging: path ~ key prefix

Simple-Module Executor (1/2)

Forking

`com.svbio.workflow.executor.invocation = forking`

- Simple-module execution ~ command-line invocation
- Command-line template: `com.svbio.workflow.executor.commandline`
  - Placeholders
    - `<classpath>`: Contains all classes necessary to start simple-module executor in separate JVM (but no more)
    - `<props>`: Contains system properties `config.file` and `com.svbio.workflow.*` (list of `–Dkey=value` arguments)
  - All other elements go through `String.format()` with two arguments
    - `Requirements#cpuCores()`
    - `Requirements#memoryGB()` times `com.svbio.workflow.executor.commandline.memscale`

```
# Default template:
commandline = [
    ${java.home}/bin/java,
    "-classpath",
    ${java.class.path},
    "<props>",
    "-Xmx%2$dm",
    com.svbio.workflow.forkedexecutor.ForkedExecutor
]
```

Simple-Module Executor (2/2)

DRMAA

```
com.svbio.workflow.executor.invocation = drmaa
```

- Simple-module execution ~ DRMAA job submission
- Command-line template like for forking executor
- Template for native arguments:
  - goes through `String.format()` with two arguments
    - `Requirements#cpuCores()`
    - `Requirements#memoryGB()` times `com.svbio.ckservice.executor.drmaa.memscale`
  - memscale should be slightly higher than for JVM

```
# Default native arguments
nativespec = "-l slots_free=%d,virtual_free=%dM"
```

lifecode

Database Logging

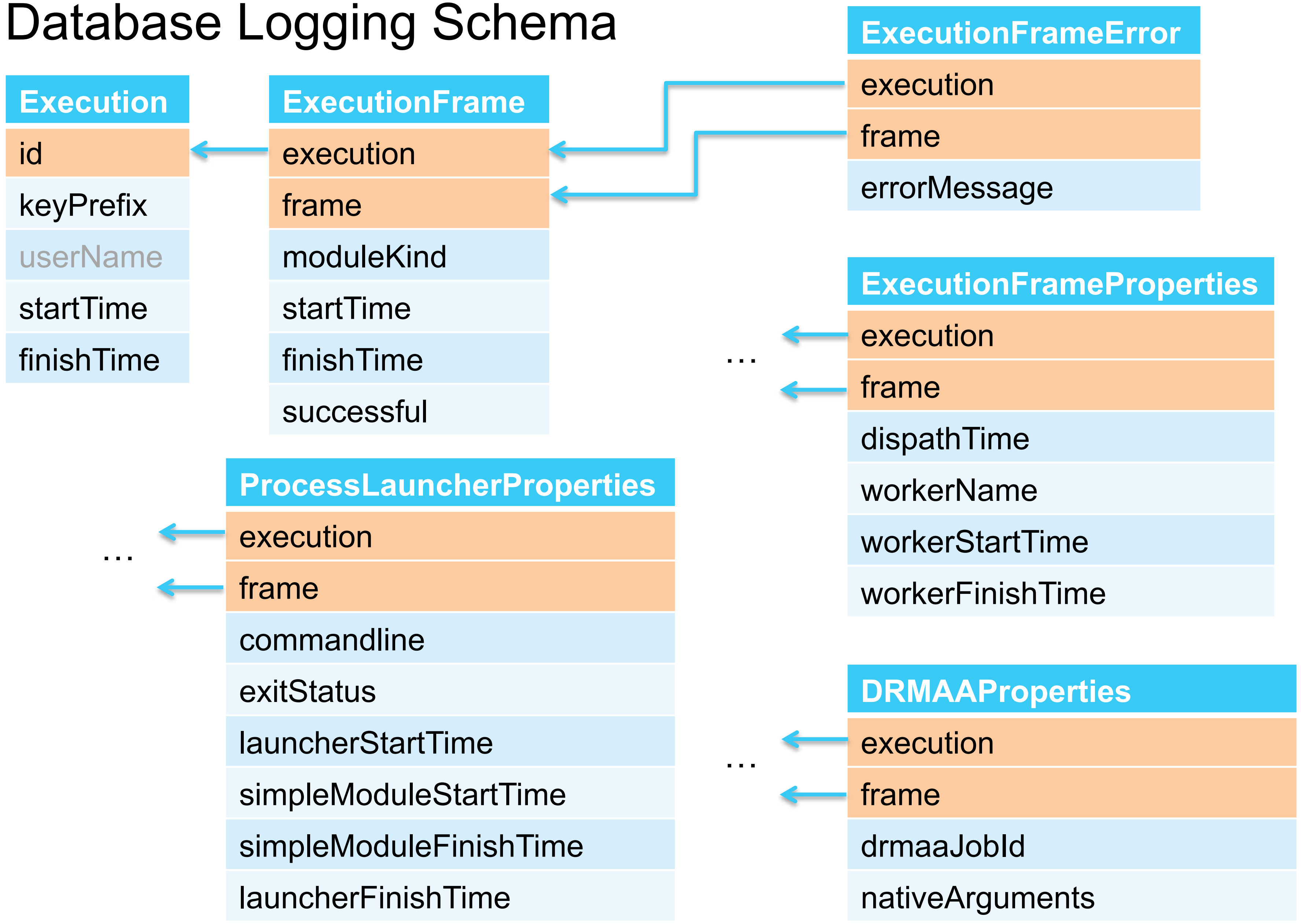
Logging of Interpreter Events

- Table structure chosen for legacy reasons (next slide)
- Schema (table qualifier) configurable
- PostgreSQL and H2 drivers included by default

```
com.svbio.workflow {
    # Settings pertaining to database logging.
    database {
        # The schema (table qualifier) in which the database tables reside.
        schema = "public"

        # Java Persistence API 2.1 properties. All properties in this group
        # will be passed to method Persistence#createEntityManagerFactory
        # as-is (without the "com.svbio.workflow.database" prefix).
        javax.persistence {
            jdbc.driver = org.h2.Driver
            jdbc.url = "jdbc:h2:mem:workflowservice"
            jdbc.user = ""
            jdbc.password = ""
            schema-generation.database.action = create
        }
    }
```

Database Logging Schema



Execution
id
keyPrefix
userName
startTime
finishTime

ExecutionFrame
execution
frame
moduleKind
startTime
finishTime
successful

ExecutionFrameError
execution
frame
errorMessage

ExecutionFrameProperties
execution
frame
dispathTime
workerName
workerStartTime
workerFinishTime

ProcessLauncherProperties
execution
frame
commandline
exitStatus
launcherStartTime
simpleModuleStartTime
simpleModuleFinishTime
launcherFinishTime

DRMAAProperties
execution
frame
drmaaJobId
nativeArguments

Example Projects

Defining a CloudKeeper Bundle

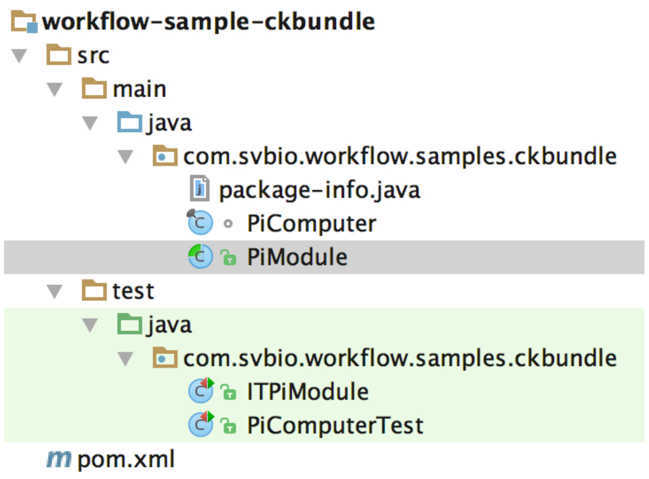
Embedding CloudKeeper

lifecode

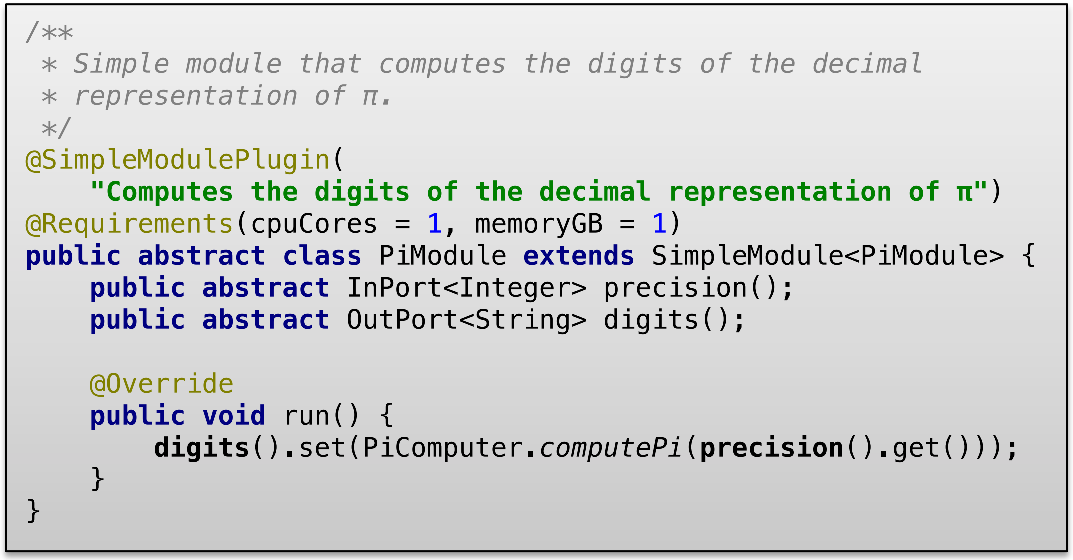
Example: CloudKeeper Bundle (1/2)

Demonstrates

- Defining simple module in CloudKeeper bundle
- CloudKeeper Maven plugin



```
workflow-sample-ckbundle
▼ src
    ▼ main
        ▼ java
            ▼ com.svbio.workflow.samples.ckbundle
                  package-info.java
                  PiComputer
                  PiModule
    ▼ test
        ▼ java
            ▼ com.svbio.workflow.samples.ckbundle
                  ITPiModule
                  PiComputerTest
  pom.xml
```



```java
/**
 * Simple module that computes the digits of the decimal
 * representation of π.
 */
@SimpleModulePlugin(
    "Computes the digits of the decimal representation of π")
@Requirements(cpuCores = 1, memoryGB = 1)
public abstract class PiModule extends SimpleModule<PiModule> {
    public abstract InPort<Integer> precision();
    public abstract OutPort<String> digits();

    @Override
    public void run() {
        digits().set(PiComputer.computePi(precision().get()));
    }
}
```

lifecode

Example: CloudKeeper Bundle (2/2)

Integration Tests

- examples corresponding to use cases
  - debugging, smoke test, real-sized tests

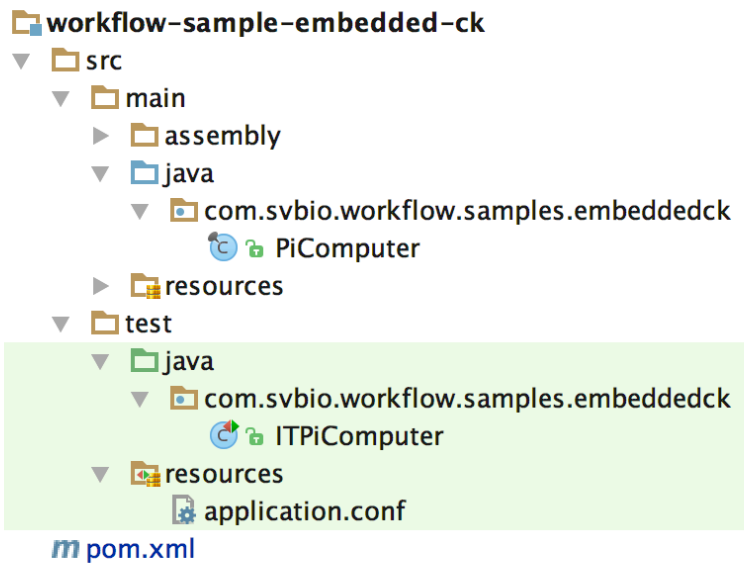
Simple-Module Executor	Staging Area	Runtime-Context Provider
in-JVM	in-memory (HashMap-based)	DSL class walker
forking	file-system	"
DRMAA	"	"

  - DSL class walker because artifacts are deployed only *after* integration-test phase

Example: Embedded CloudKeeper (1/2)

Run CloudKeeper with fixed workflow

- Example: Compute π
- Uses Maven runtime-context provider
  - No Maven dependency on compute-pi module
  - Current JVM: no dynamic loading of Java classes



```
workflow-sample-embedded-ck
▼ src
    ▼ main
        ▶ assembly
        ▼ java
            ▼ com.svbio.workflow.samples.embeddedck
                PiComputer
        ▶ resources
    ▼ test
        ▼ java
            ▼ com.svbio.workflow.samples.embeddedck
                ITPiComputer
        ▼ resources
            application.conf
  pom.xml
```

```java
RuntimeContextComponent runtimeContextComponent
    = DaggerRuntimeContextComponent.builder()
        .configModule(new ConfigModule(config))
        .lifecycleManagerModule(new LifecycleManagerModule(lifecycleManager))
        .mavenRuntimeContextFactoryModule(new MavenRuntimeContextFactoryModule(
            ClassLoader.getSystemClassLoader()
        ))
        .build();
```

  - Forked JVM: dynamically created class loader

lifecode

Example: Embedded CloudKeeper (2/2)

CloudKeeper API for Inputs and Outputs

```java
MutableModule<?> module = new MutableProxyModule()
    .setDeclaration("com.svbio.test.PiModule");

WorkflowExecution workflowExecution = cloudKeeperEnvironment
    .newWorkflowExecutionBuilder(module)
    .setInputs(Collections.singletonMap(
        SimpleName.identifier("precision"), precision)
    )
    .setBundleIdentifiers(Collections.singletonList(Bundles.bundleIdentifierFromMaven(
        "com.svbio.workflow.samples",
        "workflow-sample-ckbundle",
        Version.valueOf("1.0.0.0-SNAPSHOT")
    )))
    .start();

String result = (String) WorkflowExecutions
    .getOutputValue(workflowExecution, "digits", 1, TimeUnit.MINUTES)
```

Running CloudKeeper Workflows from Scripts

Groovy

- Script language on top of the JVM
- Automatic Maven dependency retrieval with `@Grab` annotations

```
@GrabResolver(
    name = 'My Artifact Repoitory',
    root = // URI of artifact repository
)
// ...
@Grab(
    group = 'com.svbio.workflow',
    module = 'workflow-service',
    version = '1.0.0.0-SNAPSHOT'
)
import // ...

def workflowExecution = cloudKeeperEnvironment
    .newWorkflowExecutionBuilder(module)
    .setInputs([(SimpleName.identifier('precision')) : 10])
    .setBundleIdentifiers([URI.create(
        'x-maven:com.svbio.workflow.samples:workflow-sample-ckbundle:' +
        'ckbundle:1.0.0.0-SNAPSHOT'
    )])
    .start();
```