Configuration

Toolset

S4 provides a set of tools to:

- define S4 clusters: `s4 newCluster`
- start S4 nodes: `s4 node`
- package applications: `s4 s4r`
- deploy applications: `s4 deploy`
- start a Zookeeper server for easy testing: `s4 zkServer`
    - `s4 zkServer -t` will start a Zookeeper server and automatically configure 2 clusters
- view the status of S4 clusters coordinated by a given Zookeeper ensemble: `s4 status`

```
./s4
```

will give you a list of available commands.

```
./s4 <command> -help
```

will provide detailed documentation for each of these commands.

Cluster configuration

Before starting S4 nodes, you must define a logical cluster by specifying:

- a name for the cluster
- a number of partitions (~ tasks)
- an initial port number for listener sockets
    - you must specify a free port, considering that each of the nodes of the cluster will open a different port, with a number monotonically increasing from the initial number. For instance, for a cluster of 10 nodes and an initial port 12000, ports 12000 to 12009 will be used among the nodes.
    - those ports are used for inter node communication.

The cluster configuration is maintained in Zookeeper, and can be set using S4 tools:

```
./s4 newCluster -c=cluster1 -nbTasks=2 -flp=12000
```

See tool documentation by typing:

```
./s4 newCluster -help
```

Node configuration

When starting an S4 node on a given host, you need to specify:

- the connection string to the cluster management system (Zookeeper)
    - localhost:2181 by default
- the name of the logical cluster to which this node will belong
- specific modules (optional)
- named configuration parameters (optional)

Example:

```
./s4 node -c=cluster1 -zk=host.domain.com
```

Modules configuration

S4 follows a modular design and uses Guice for defining modules and injecting dependencies.

An S4 node is composed of:

- a communication module that specifies communication protocols, event listeners and senders

- a core module that specifies the deployment mechanism, serialization mechanism

We provide default modules, but you may directly specify others through the command line, and it is also possible to override them with new modules and even specify new ones (custom modules classes must provide an empty no-args constructor). For instance, if you want to enable checkpointing, you should pass a checkpointing module (*emc* stands for extra modules classes):

```
./s4 node -c=cluster1 -emc=org.apache.s4.core.ft.FileSystemBackendCheckpointingModule
```

Parameters overriding

You will probably need to tweak default parameters of the communication layer.

You may use inline parameters:

```
./s4 node -c=cluster1 -p=param1=value1,param2=value2
```

File-based configuration

Instead of specifying node parameters inline, you may refer to a file with the '@' notation:

```
./s4 node @/path/to/config/file
```

With contents of the referenced file like:

```
-c=cluster1
-zk=host.domain.com:42000
-p=param1=value1,param2=value2
```

Logging

S4 uses logback, and here is the default configuration file. You may tweak this configuration by adding your own logback.xml file in the `lib/` directory (for a binary release) or in the `subprojects/s4-tools/build/install/s4-tools/lib/` directory (for a source release or checkout from git).

Application configuration

A simple way to pass parameters to your application is by:

- injecting them in the application class:
  ```
  @Inject
  @Named('myParam')
  param
  ```
- specifying the parameter value at node startup (using -p inline with the node command, or with the '@' syntax)

This uses an internal Guice module that automatically injects your application parameters to matching `@Named` parameters.