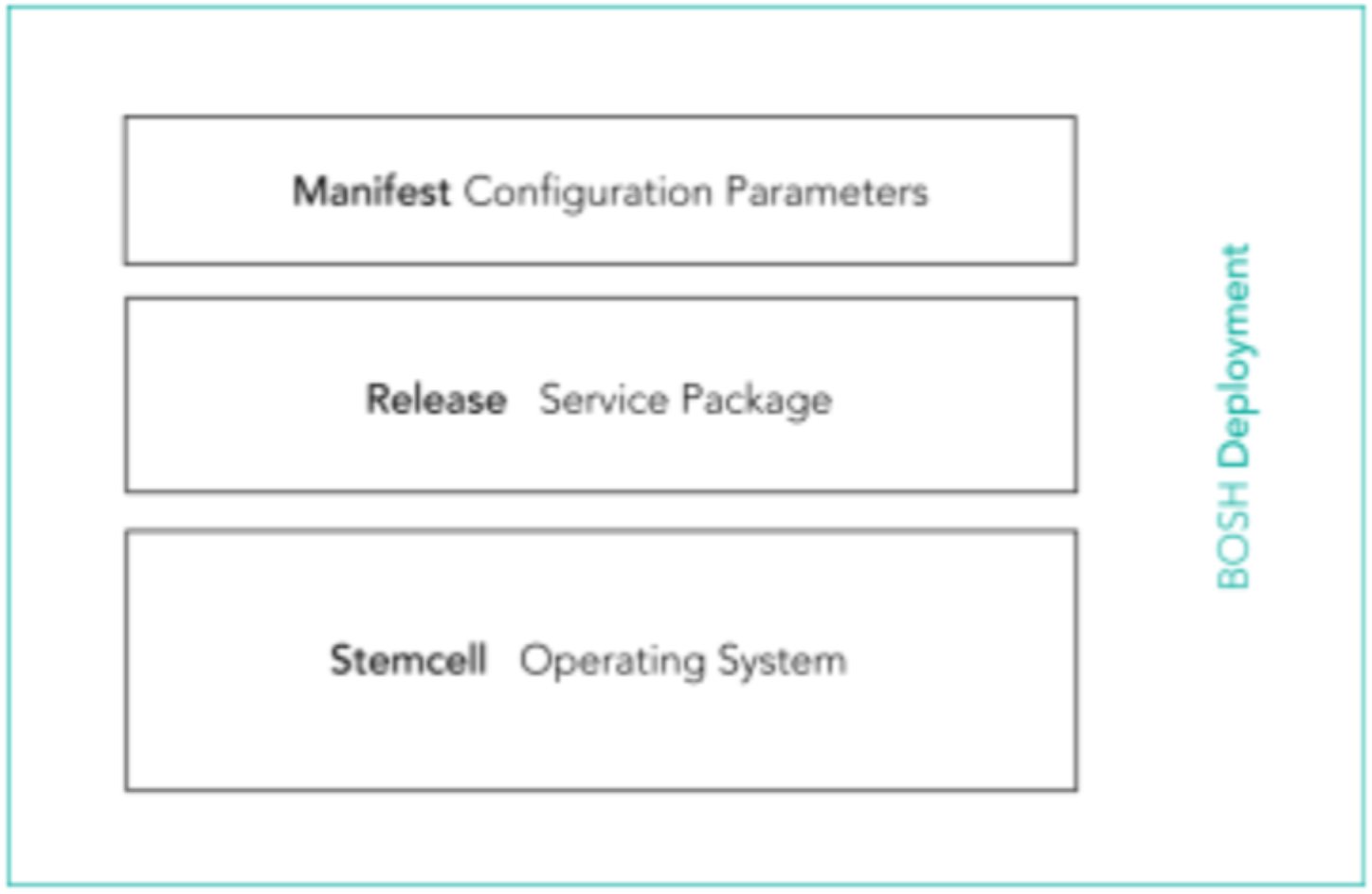
# How to build Bosh add-on?

In this bosh add-on example, we will take springboot app (<https://github.com/jigsheth57/fe-time-tracking>) and make a bosh add-on out of it.

<https://s3.amazonaws.com/pcf-softwares-57/fe-time-tracking-1.0.0.jar>

## Prerequisite

* Require bosh2 cli .. <https://bosh.io/docs/cli-v2.html#install>
* Working PCF environment

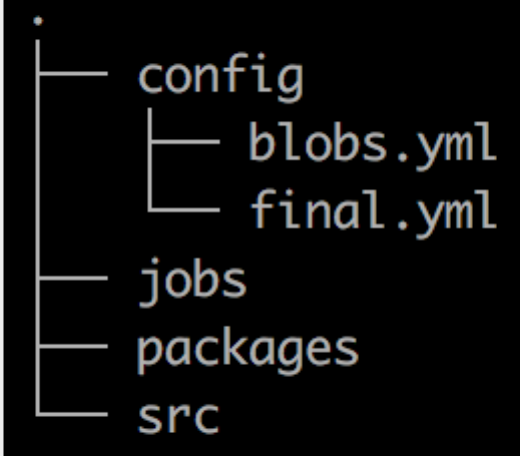


## Step 1: Create the release directory

**$ bosh2 init-release --dir sample-boshrelease**

Note: this will create directory structure in working directory for sample-boshrelease.

When deploying your release, BOSH places compiled code and other resources in the **/var/vcap/** directory, which BOSH creates on the job VMs. The four directories you just created, jobs, packages, src, and blobs, appear on job VMs as **/var/vcap/jobs, /var/vcap/packages, /var/vcap/src,** and **/var/vcap/blobs**, respectively.

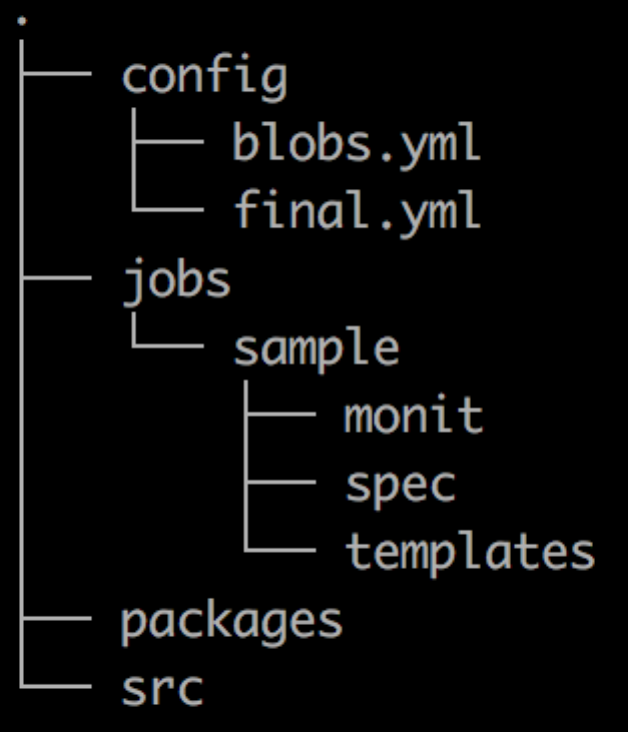


## Step 2: Create Job stub

**$ cd sample-boshrelease/**

**$ bosh2 generate-job sample**

Note: this will create directory structure and stub files under ./jobs directory for job called “sample”.



### Step 2.1 Create control scripts for job

Every job needs a way to start and stop. You provide that by writing a control script and updating the monit file.

The control script:

* Includes a start command and a stop command.
* Is an ERB template stored in the templates directory for the relevant job.

For each job, create a control script that configures the job to store logs in /var/vcap/sys/log/JOB\_NAME. Save this script as **ctl.erb**in the templates directory for its job.

**$ vi jobs/sample/templates/ctl.erb**

Sample job control script:

|  |
| --- |
| #!/bin/bash  set +e  set -xv  RUN\_DIR=/var/vcap/sys/run/sample  LOG\_DIR=/var/vcap/sys/log/sample  PIDFILE=${RUN\_DIR}/pid  export DEBUG\_LOG=$LOG\_DIR/sample.log  touch $DEBUG\_LOG  chmod -R 755 /var/vcap/sys  chmod 777 $DEBUG\_LOG  function log\_debug() {  echo `date` $1 >> $DEBUG\_LOG  }  log\_debug "Kicking off ctl script as `whoami` with $1"  case $1 in  start)  mkdir -p $RUN\_DIR $LOG\_DIR  chown -R vcap:vcap $RUN\_DIR $LOG\_DIR  echo $$ > $PIDFILE  export OPENJDK\_HOME=/var/vcap/packages/openjdk/  export SAMPLE\_PACKAGE=/var/vcap/packages/sample/  pushd ${OPENJDK\_HOME}  if [[ -d bin && `bin/java -Xmx64M -version 2>&1` =~ "OpenJDK 64-Bit Server" && $? == 0 ]]; then  export JAVA\_BINARY=${OPENJDK\_HOME}bin/java  cd ..  else  log\_debug "JVM is not properly packaged"  exit 1  fi  log\_debug "Located Java Binary at: ${JAVA\_BINARY}"  exec $JAVA\_BINARY -jar $SAMPLE\_PACKAGE/fe-time-tracking-\*.jar -Dserver.port=<%= properties.sample.listen\_port %> \  >> $LOG\_DIR/sample.stdout.log \  2>> $LOG\_DIR/sample.stderr.log  ;;  stop)  kill -9 `cat $PIDFILE`  rm -f $PIDFILE  ;;  \*)  echo "Usage: ctl {start|stop}" ;;  esac |

### Step 2.2 Update job monit file

The monit file:

* Specifies the process ID (pid) file for the job
* References each command provided by the templates for the job
* Specifies that the job belongs to the vcap group

**$ vi jobs/sample/monit**

|  |
| --- |
| check process sample  with pidfile /var/vcap/sys/run/sample/pid  start program "/var/vcap/jobs/sample/bin/ctl start"  stop program "/var/vcap/jobs/sample/bin/ctl stop"  group vcap |

### Step 2.3 Update job specs

At compile time, BOSH transforms templates into files, which it then replicates on the job VMs.

The template names and file paths are among the metadata for each job that resides in the job spec file.

In the job spec file, the templates block contains key/value pairs where:

* Each key is template name
* Each value is the path to the corresponding file on a job VM

The file paths that you provide for templates are relative to the /var/vcap/jobs/<job\_name> directory on the VM. For example, bin/ctl becomes /var/vcap/jobs/<job\_name>/bin/ctl on the job VM. Using bin as the directory where these files go is a convention.

**$ vi jobs/sample/spec**

|  |
| --- |
| ---  name: sample  templates:  ctl.erb: bin/ctl  packages:  - openjdk  - sample  properties:  sample.listen\_port:  description: "Listen Port of the Sample instance"  default: 8080 |

## Step 3: Add Openjdk dependency package

Packages give BOSH the information needed to prepare the binaries and dependencies for your jobs.

**$ bosh2 generate-package openjdk**

### Step 3.1: Update packing specs

**$ vi packages/openjdk/spec**

|  |
| --- |
| ---  name: openjdk  dependencies: []  files:  - openjdk-1.8.0\_144.tar.gz # From https://s3.amazonaws.com/pcf-softwares-57/openjdk-1.8.0\_144.tar.gz |

### Step 3.2: Create packaging scripts

Write packaging scripts to instruct BOSH how to do this. The instructions may involve some combination of copying, compilation, and related procedures.

**$ vi packages/openjdk/packaging**

|  |
| --- |
| #!/bin/bash  # abort script on any command that exit with a non zero value  set -e -x  archive=`echo openjdk-\*tar.gz`  if [[ -f $archive ]] ; then  echo "Archive found"  else  echo "Archive not found"  exit 1  fi  cp -a $archive ${BOSH\_INSTALL\_TARGET}  export OPENJDK\_HOME=/var/vcap/packages/openjdk  pushd ${OPENJDK\_HOME}  archive=`ls open\*gz`  tar zxvf $archive |

## Step 4: Add Sample (springboot app) dependency package

Packages give BOSH the information needed to prepare the binaries and dependencies for your jobs.

**$ bosh2 generate-package sample**

### Step 3.1: Update packing specs

**$ vi packages/sample/spec**

|  |
| --- |
| ---  name: sample  dependencies: []  files:  - fe-time-tracking-1.0.0.jar # From https://s3.amazonaws.com/pcf-softwares-57/fe-time-tracking-1.0.0.jar |

### Step 3.2: Create packaging scripts

Write packaging scripts to instruct BOSH how to do this. The instructions may involve some combination of copying, compilation, and related procedures.

**$ vi packages/sample/packaging**

|  |
| --- |
| #!/bin/bash  # abort script on any command that exit with a non zero value  set -e -x  archive=`echo sample/fe-time-tracking-\*jar`  if [[ -f $archive ]] ; then  echo "Archive found"  else  echo "Archive not found"  exit 1  fi  cp -r $archive ${BOSH\_INSTALL\_TARGET} |

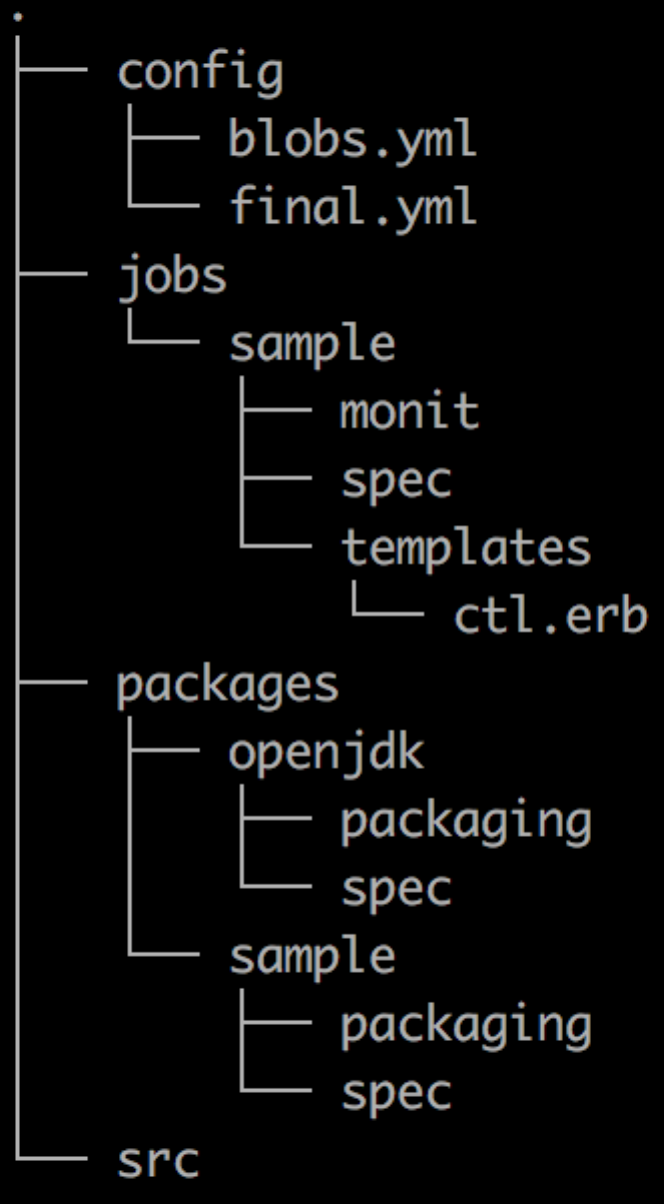
## Step: 5 Add blobs from packaging

This step allows you to add binaries packages to bosh release.

Note: The local type blobstore is suitable for learning but the resulting release cannot be shared. For that reason, you should configure a non-local, publicly available blobstore for releases that you intend to share.

**$ vi config/final.yml**

|  |
| --- |
| ---  blobstore:  provider: local  options:  blobstore\_path: /tmp/sample-release-blobs  name: sample-release |



Download the binaries packages from S3 to your local machine.

**$ wget** [**https://s3.amazonaws.com/pcf-softwares-57/fe-time-tracking-1.0.0.jar**](https://s3.amazonaws.com/pcf-softwares-57/fe-time-tracking-1.0.0.jar)

**$ wget** [**https://s3.amazonaws.com/pcf-softwares-57/openjdk-1.8.0\_144.tar.gz**](https://s3.amazonaws.com/pcf-softwares-57/openjdk-1.8.0_144.tar.gz)

Add blobs to the bosh release

**$ bosh2 add-blob <download path>/openjdk-1.8.0\_144.tar.gz openjdk-1.8.0\_144.tar.gz**

**$ bosh2 add-blob <download path>/fe-time-tracking-1.0.0.jar fe-time-tracking-1.0.0.jar**

Upload blobs to the bosh release

**$ bosh2 upload-blobs**

## Step 6: Create Bosh release

Create bosh release

**$ bosh2 create-release --final --version=1.1.0 --tarball=<path to save release>/sample-boshrelease-1.1.0.tgz**