The road to sbt 1.0

Paved with Server
Who are we?

@eed3si9n
(Eugene Yokota)
- sbt core dev
- scalaxb
- treehugger.scala
- "learning Scalaz"
- sbt-assembly etc

Josh Suereth
- Big Nerd
- Leader of tools team @ Typesafe
- Author
  - Scala in Depth
  - sbt in Action
Agenda

● A brief *(incomplete and mostly wrong)* history of Build Tools
● A roadmap to sbt 1.0
  ○ Modularization
● sbt-server
A brief history of build tools

Incomplete and mostly wrong
$ make -j 2 hello

# Makefile
OBJ = test.o test2.o
%.o : %.c
  $(CC) -c -o $@ < $<

hello : $(OBJ)
  gcc -o $@ $^
Q: How do I build this program?

- new! Dependency-based programming
- Uses system command to carry out tasks
- problem! Machine-dependent
Q: How do I build on *this machine*?

- Generate configure script and Makefile templates
  - Configure generates header scripts and real makefile
  - Configure discovers what the machine has
- Use an even MORE arcane syntax
  - m4 macros
  - magic strings
Internet
$ ant dist

<project name="MyProject" default="dist" basedir="."/>

<description>example build file</description>

<property name="src" location="src"/>

<property name="build" location="build"/>

<property name="dist" location="dist"/>

<target name="compile" description="compile the source ">
  <javac srcdir="${src}" destdir="${build}"/>
</target>

<target name="dist" depends="compile"
  description="generate the distribution">
  <mkdir dir="${dist}/lib"/>
  <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar"
      basedir="${build}"/>
</target>

</project>
Q: How do I build on *this machine*?

- **new!** Built-in tasks like *mkdir*, *javac*, and *jar*
  - Self-contained and platform-independent
- task plugins (like ant-contrib)
- **problem!** Hard to reuse build logic
Package managers

Q: How do I find library source/binary?

- **new!** Metadata to track library dependencies
- **new!** Repository to host source and binary
Maven (2004)

Q: How do I find library source/binary?
Q: How do I enforce consistent builds?

- Metadata to track library dependencies
- Repository to host source and binary
- **new!** Default build (Convention over Configuration)
- **new!** Plugins that allowed reuse of build logic
- **problem!** Difficult to customize
Q: How do I find library source-binary?

- Metadata to track library dependencies
- Repository to host source and binary
- Maven integration on top of Ant
rule '.o' => ['.c'] do |t|
  sh "cc #{t.source} -c -o #{t.name}"
end

task :name, [:first_name, :last_name] do |t, args|
  puts "First name is #{args.first_name}"
  puts "Last name is #{args.last_name}"
end
Q: How do I build on this machine?
Q: How do I customize my build?

- **new!** Internal DSL
  - Leverage Ruby libraries rather than shell programs
  - Provide cross-platform ruby libraries for supported language tools
- Real programming language to write tasks
O NOES XML!

SAX parsing is declared uncool
(immediately after inception)

REST/JSON attempt to displace SOAP
Q: How do I customize my build?

- Use an internal DSL and a real program language

Q: How do I find library source/binary?

- Maven integration
Q: How do I find library source/binary?

Q: How do I customize my build?

Q: How can I develop faster?

- **new!** Incremental compiler tracking source deps
- **new!** Interactive shell + library-aware REPL
- **new!** Test framework abstraction
- **new!** Parallel by default
- **shabby-chic!** ANSI colors
- Internal DSL + Maven integration
Google Blaze (2009)

Q: How can I develop faster?
Q: How do I avoid version conflicts?

- new! Incremental build
- new! Cache builds remotely
- new! Clustered Building
- Abandon Maven. Check in all source to version control. Force everyone on the same version.
- 1 SCM repo for all projects
- Assume homogeneous-ish environment
Q: How can I develop faster?
Q: How do I avoid version conflicts?

- Incremental build
- Cache builds remotely
- Force all dependencies to the same version for all projects
- Put everything in a mono-repository
History

- **make** (1977)
- **automake** (1996)/**autoconf** (1991)
- **RPM** (1997)
- **Ant** (2000)
- **Maven** (2004)
- **Ivy** (2004)
- **sbt** (2008)
- **Buildr** (2008)
- **Gradle** (2009)
- **Leiningen** (2009)
- **Blaze** (2009)
- **Pants** (2014)
- **Buck** (2014)

History Recap

- **sbt has inherited:**
  - dependency based programming (Make)
  - internal build DSL (Rake)
  - Package/Library Management (Maven)
  - Convention over configuration (Maven)
  - Re-usable build flow (Maven)

- **sbt brings:**
  - interactivity (on the shell)
  - parallel by default
A roadmap to sbt 1.0

stability + sbt server => modularization
Stability

1. Conceptual stability
2. Binary compatibility of sbt plugins
3. Source compatibility of build files
Concepts (mostly stable)

1. Scala incremental compiler
2. Dependency manager (Scala-aware)
3. Task and plugin system (using Scala)
4. Test framework abstraction
5. Text-based interactive shell

5. sbt server + client(s)
Plugin binary compatibility

- sbt 0.13 maintained 18 months of bincompat
  - Lots of hacks and effort. Unable to remove cruft.
- sbt 1.x.y should be bincompat with 1.0.0
- Need to minimize surface API
  - able to add small features when requested
Build source compatibility

- Source compatibility of your build.sbt
- sbt 1.x.y should be stable
- sbt 1.0 gives us opportunity to break DSL
  - Deprecate project/build.scala?
  - Unify sbt shell key syntax w/ Scala DSL syntax
Modularization

Componentize stable features, innovate new ideas

1. Pull out cohesive subprojects
2. Distinguish public API and internal details
3. Document usages
4. Clean up historical code
5. Cross publish for latest scala versions 
   (if applicable)
6. Publish to Maven Central (or JCenter)
Module candidates

- Launcher API
- Serialization API
- Compiler/REPL API
- Test framework API
- Dependency Management API
- Network API
- IO API
- Task DSL
- Completion API
- sbt client (sbt-server)
sbt-server

Architecture and Design
The problem

- Many things want access to the build
- We need to centrally control build-related tasks to avoid breakage.
  - IntelliJ auto-import + "sbt ~ test"
  - Activator + "play run"
sbt-server design

- sbt client
- commands / watches
- disk (target/)
- sbt-server
- changes
- Events (logs, status, custom)
Before server - Command Engine

- Terminal Client

  - CMD (compile)
    - Engine (sequential)
      - Next command

  - Read next command from stdin
  - Read previous log file
  - Reload the build
  - Task Engine (parallel)
Problem: Server discovery

Who starts the sbt-server, how do we know it is running, how do we connect?
sbt-launcher

- sbt 0.13.2 launcher had support for "servers"
  - launch config specifies "lock" file which prevents more than one service being started
  - "lock" file contains location (ip / port) of the service
  - server is tested (ping) for liveness by the launcher.
Problem: Disconnects

- Client may not be the one to start sbt
- Client may disconnect from server, or server may crash
val connector = SbtConnector("terminal", "Command Line Terminal", configuration.baseDirectory)

def onConnect(client: SbtClient): Unit = {
  client handleEvents { … }
  client watch …
}

connector.open(onConnect, onError)(<execution context>)
Connect as an Event

```scala
val connector = SbtConnector(
  "terminal", "Command Line Terminal",
  configuration.baseDirectory)

def onConnect(client: SbtClient): Unit = {
  client handleEvents { … }
  client watch …
}

connector.open(onConnect, onError)(<execution context>)
```

Clients reconstruct their watches and restore their view of the build on any reconnect.
Problem: protocol

We needed a good mechanism for client + server to communicate.

- Something that can evolve and maintain compatibility
- Something that is easy for plugin authors to extend.
sbt-serialization

- New library based on Scala Pickling
  - JSON - currently supported format.
- sbt-remote-control has versioned protocol and tests.
- (coming) plugin *Keys as protocol
  - remote clients can watch/get values
  - require pickler/unpickler when creating *Key
  - ??? release plugin "client" jar separate from impl
Problem: Input

sbt tasks may want the user to type input
client.requestExecution(
    "run",
    Some(TerminalInteraction -> TerminalThread)
)
Example task (input)

readInput := {
val context = (interactionService in Global).value
context.readLine("> ", mask=false)
}

Problem: commands block

When running the `run` task, all other build commands are blocked.
Background Jobs

- tasks can "fork" background jobs in server
- clients can discover/connect/stop with background jobs.
Input/Interaction Design

● Commands
  ○ The requesting client provides the means for terminal interaction
  ○ Logs and stderr/stdout are sent to all clients.

● Background Jobs *(Not Completed Yet)*
  ○ Any client can take terminal interaction for a background task
  ○ opt-in for getting stdout/stderr events
Problem: Existing plugins

Current plugins + sbt should be able to try out sbt server, without breaking anything
sbt-core-next

A new, optional, plugin for sbt 0.13.x series which provides the new services for sbt-server

- BackgroundRunService
- InteractionServicePlugin
- SendEventServicePlugin
- SerializersService

In use in Play 2.4
What's next?
sbt-server TODOS

- **Interaction Improvements**
  - readline abstraction (a.k.a. Scala REPL support)
  - Background Job hooks

- **meta-project as first class citizen**
  - replace `reload plugins` command for first-class support
  - `~` as first-class citizen in server

- **kill-on-bad-state watchdog**

- **sbt-terminal-client (drop existing client)**
How can I help?

http://www.scala-sbt.org/community.html#how-can-I-help

- Follow @scala_sbt
- Contribute to StackOverflow sbt tag
- Report bugs
- Create plugins
  - Try/migrate your plugins to remote APIs: https://github.com/sbt/sbt-core-next
- Patch the core
  - Github Community labeled issues
  - Subscribe to sbt-dev list
  - Join the discussion on Gitter sbt/sbt