Desktop SDK

Solution Architect Training

Jonathan Hsieh, with feedback and help from Team Desktop
Cloudera, Inc
3/17/2010

Cloudera Desktop SDK

- Cloudera Desktop
  - A **web application platform** that uses the **desktop metaphor** to provide a **single interface** and **consistent user experience** to monitor and control an enterprise's cluster resources.
- The Cloudera Desktop Software Development Kit (SDK)
  - A set of tools and framework APIs for developing and installing custom Cloudera Desktop Apps
  - It takes advantage of the extensible modular architecture of the Cloudera Desktop

Design Principles

- Web application
  - Use standard web applications security mechanisms
- Client pull model
  - URLs requests are function calls, web pages are results
- Model View Controller-ish.
  - Separation of presentation, navigation and model generation concerns.

Context

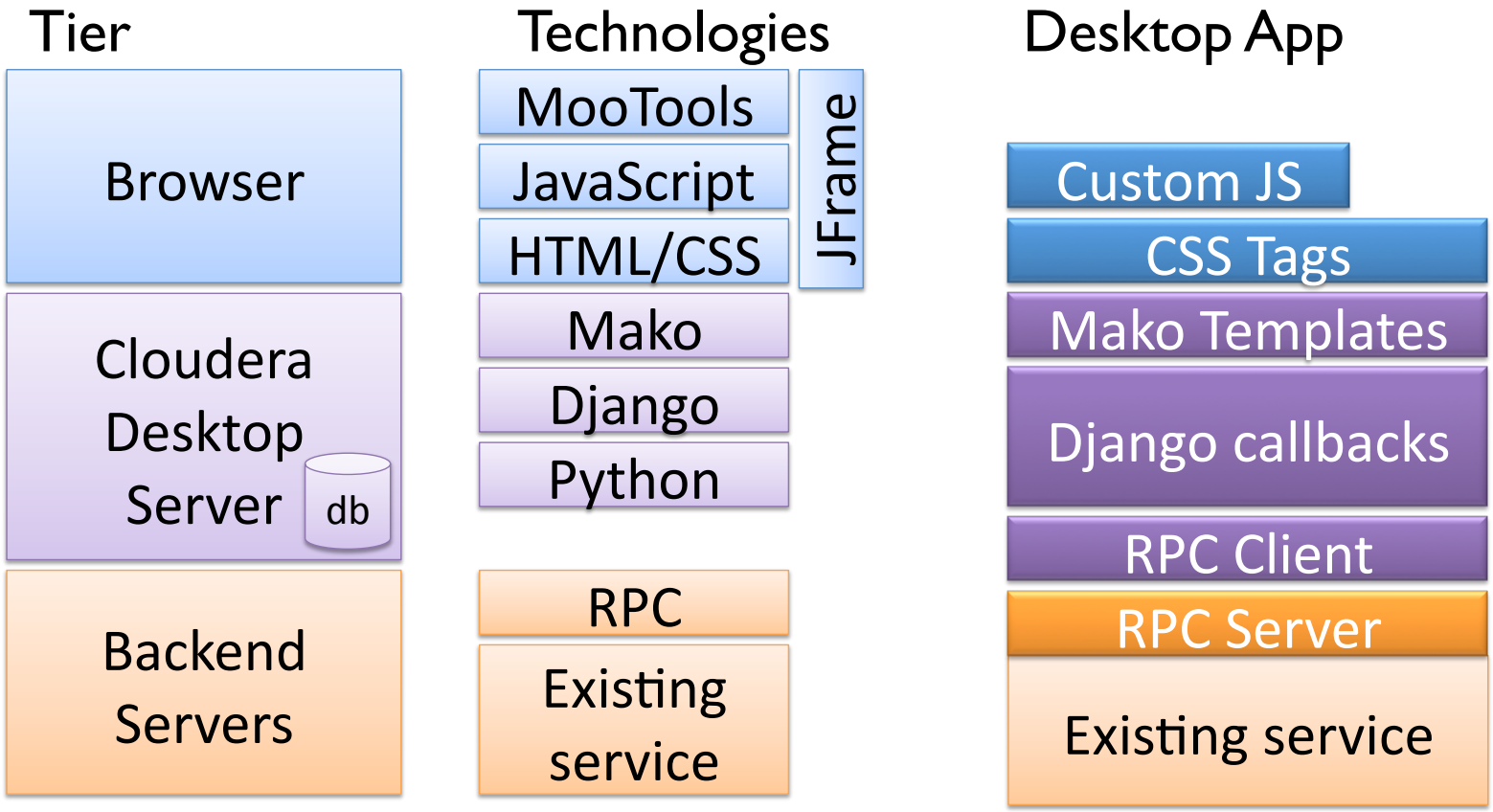
- This slide deck compliments the SDK documentation supplied by the Desktop team. To get the SDK documentation, follow these steps:
  - Clone http://git.sf.cloudera.com/index.cgi/desktop.git/
  - Go to docs/sdk
  - Compile the sdk.md (markdown) file, or read it in its original form
  - Or view the doc here: http://git.sf.cloudera.com/index.cgi/desktop.git/tree/docs/sdk/sdk.md
- This deck provides an overview of the components of a Cloudera Desktop app and how they interact. Start here and then consult the Cloudera Desktop SDK documentation for more details and stepping through building and installing an app.

Outline

- What is the Cloudera Desktop SDK?
- Anatomy of a Cloudera Desktop App
- Example Services
- Limitations

ANATOMY OF A CLOUDERA DESKTOP APP

Architecture and Technology Stack

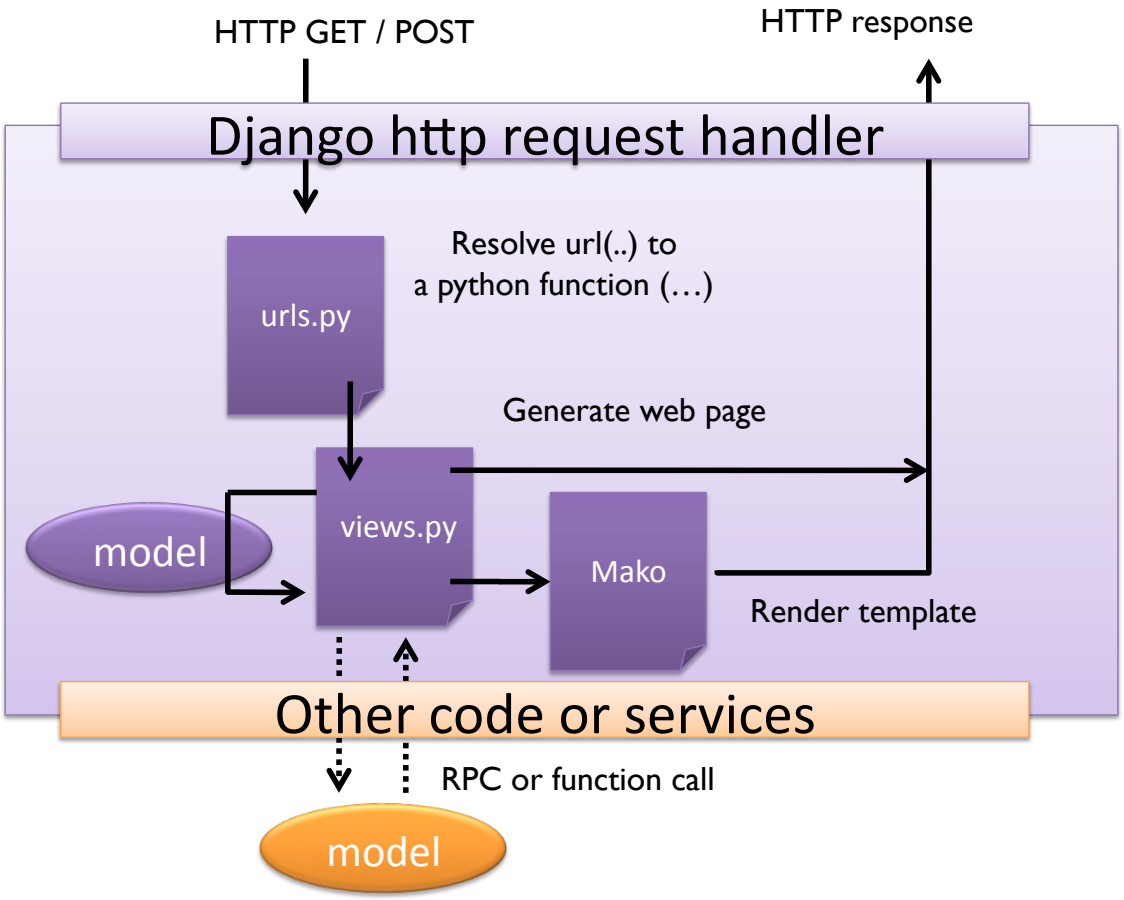


Anatomy of a Cloudera Desktop App

- A trace through a desktop request:
  - Controller
    - Django / Python. Django calls these things views.
  - View
    - HTML/CSS/Javascript via MooTools and Mako
    - Cloudera Desktop Application Tags (Jframe/CCS tags)
  - Model
    - Python. Could be a python program.
    - RPC. Could be some existing service (hadoop TT, NN, Hive, Flume, etc)

- This deck gives an overview, look at lib's doc for more specifics

Django Request Cliff's Notes



- Views.py
  - Python functions that generate web page responses
- Urls.py
  - URL translation into python function calls
- Forms.py
  - Server-side HTML Form validation

Mako Templating Cliff's Notes

- A template language for generating HTML / CSS code.
  - Design a web page
  - Embed tags that are passed in from python
  - Embed python code snippets
- Call django view funtion to render

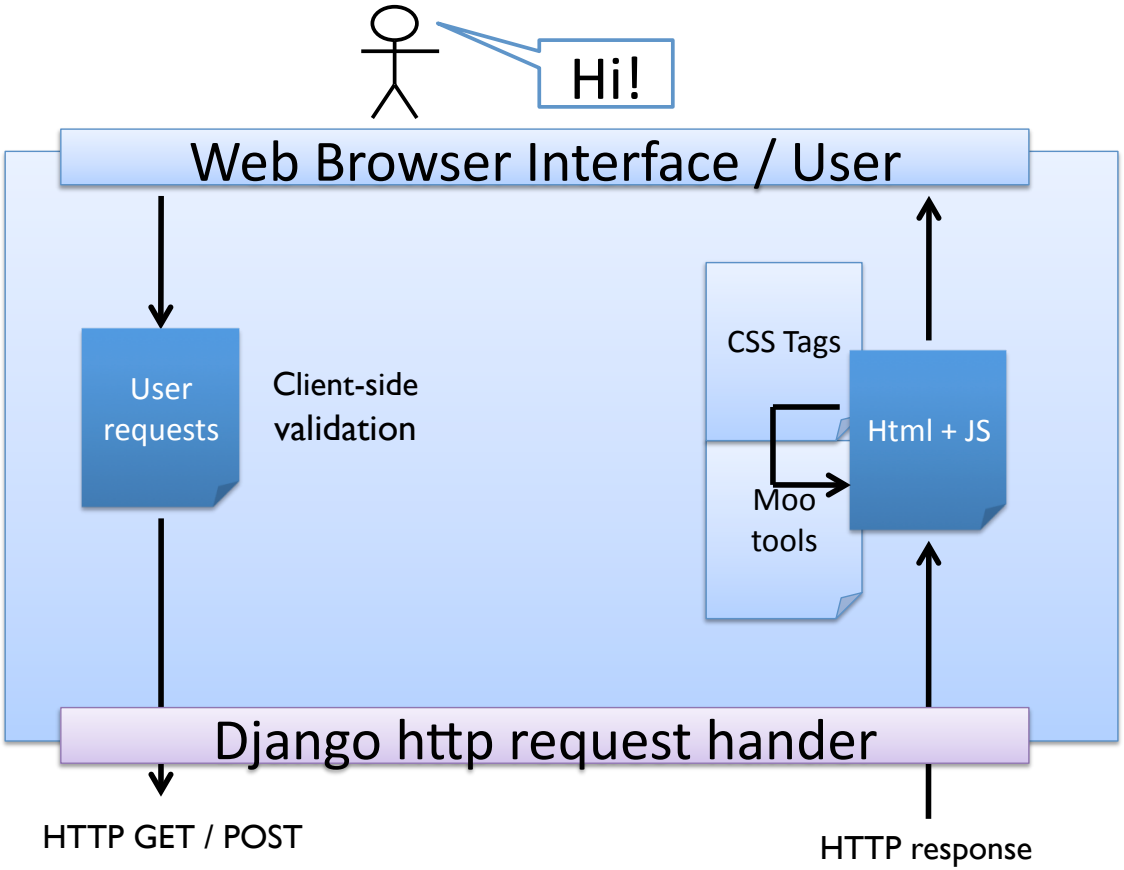
```
<!-
./src/app/templates/index.mako
-->

<html>
<head>
  <title>App</title>
</head>
<body>
  ${status}
  ${configs}
  ${commands}
</body>
</html>
```

```
#!/usr/bin/env python
# ./src/app/views.py
from desktop.lib.django_util import render

def index(request):
  (status, configs, commands, date) = …
  return render('index.mako', request, dict
    (status=status, configs=configs,
    commands=commands,date=datetime.datetime.now()))
```

Presentation Layer Cliff's Notes



- Embed web app in Cloudera Desktop **Jframe**
  - Behind the scenes the Mootools tool kit for JavaScript enriches the behavior based on class tags

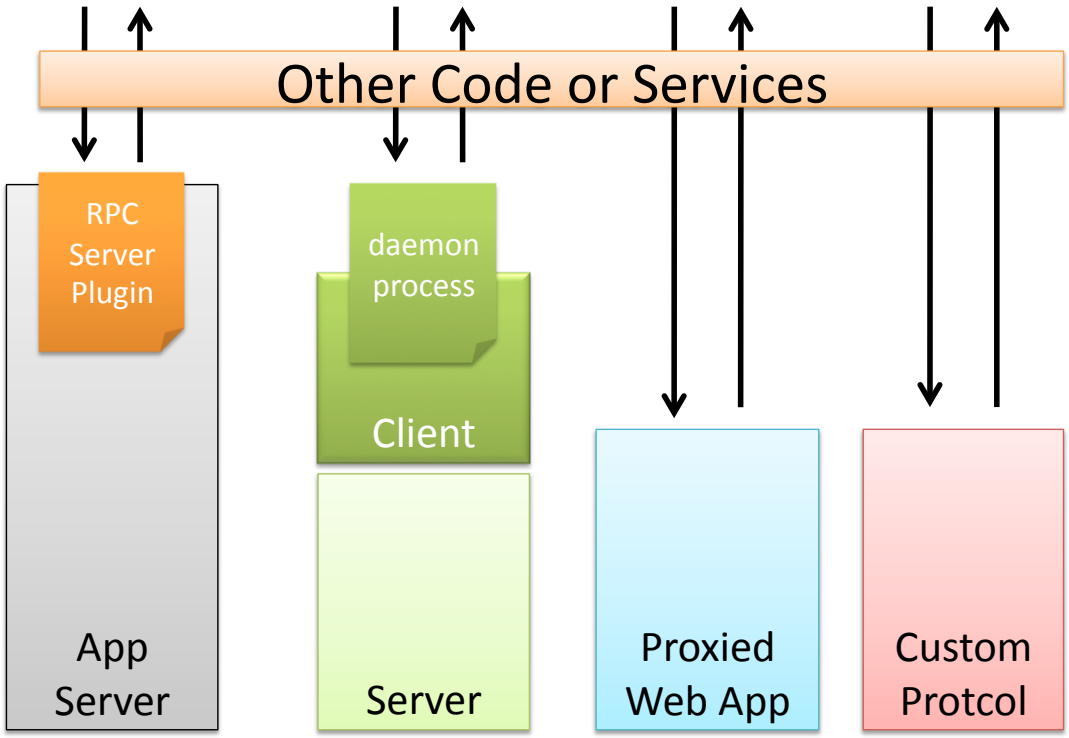
.

Presentation Styling Cliff's Notes

- Develop web app outside of desktop and just embed it
  - Add Cloudera Desktop CSS tags
  - Add specific CSS class tags for styles and behaviors
  - JavaScript and MooTools magic!

```
<!-- proj/src/flume/templates/index.mako -->
...
  <body>
    <div class="splitview resizable">
      <div class="left_col jframe_padded">
...
      </div>
      <div class="right_col jframe_padded">
        <div class="ccs-tab_ui">
...
        </div>
      </div>
    </div>
</body>
</html>
```

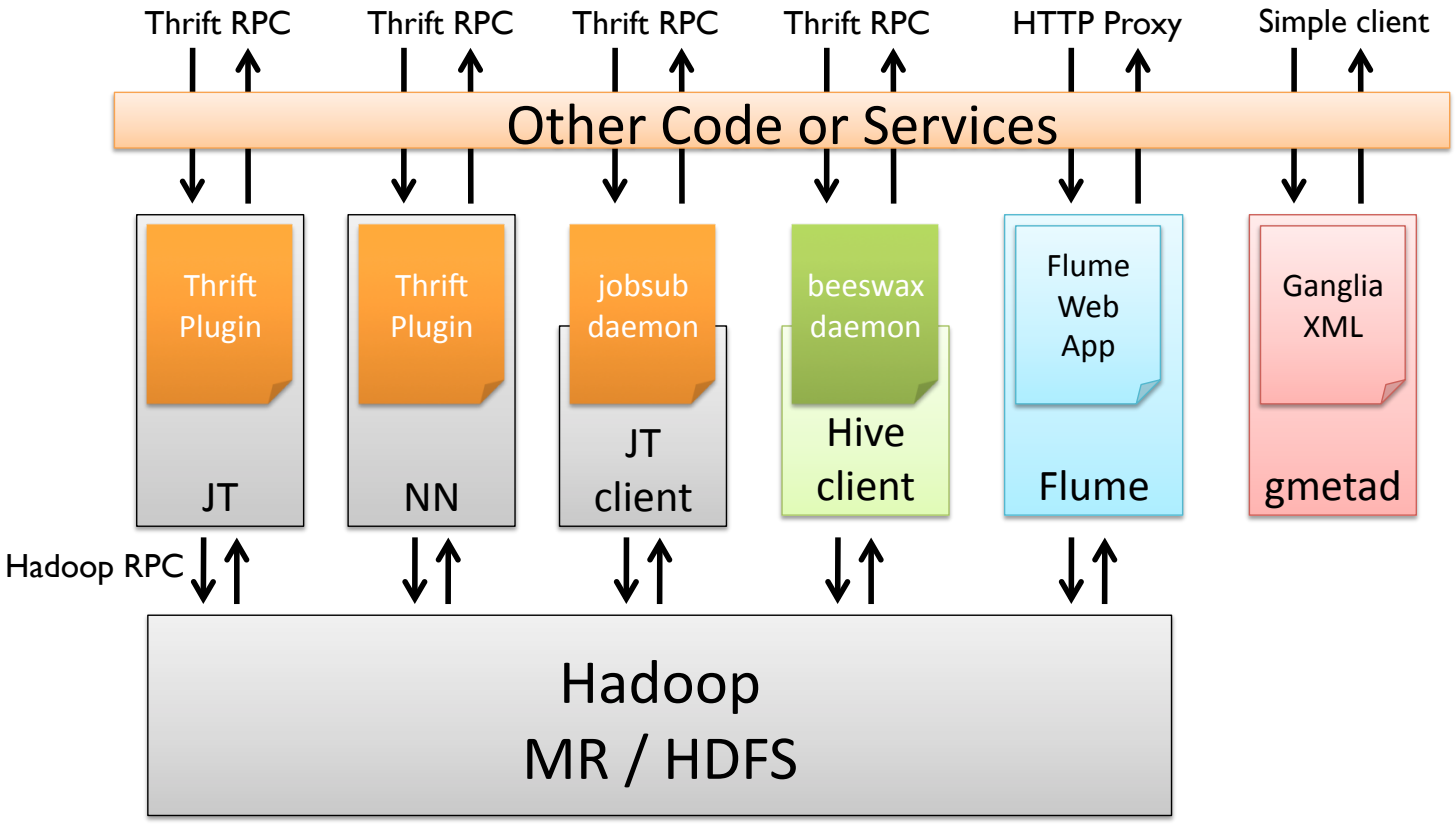
Backend Service Architecture Cliff's notes



- Backend can be in python with the Django
- Backend can be reached via RPC calls.
  - SOAP
  - XMLRPC
  - Avro
  - Thrift
  - Raw Sockets

EXAMPLE SERVICES

Example Application Backend Implementations



LIMITATIONS

Limitations

- Lack of Isolation
  - One bad desktop app can kill/hog the entire desktop server
  - One bad javascript app can kill the entire desktop
- Pull architecture
  - Currently no framework for push notifications from backend to the front end (Comet)
  - Ex: streaming console data to desktop from backend requires desktop server to poll
- Integration with javascript presentation layer libraries
  - Must be compatible with mootools. (jquery/prototype often conflicts)
- Still need to understand web programming to build sophisticated user interfaces.
- State sharing limitations due to django threading limitations.
  - No guarantees of state consistency between multiple requests containers
- Scalability
  - Requesting a ton of data will retrieve a ton of data.
  - Browser javascript engines can choke.
  - Must implement your own paging mechanisms.