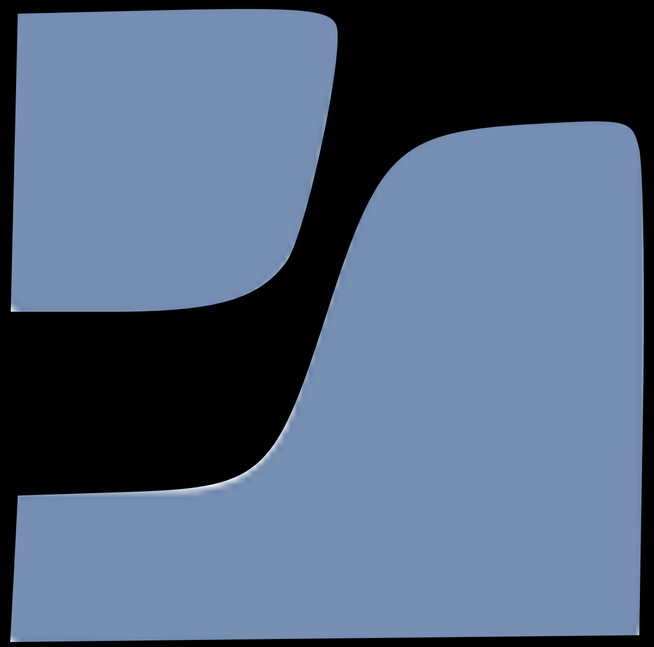
Leveraging the Jamf Pro API



Mike Matz
Systems Engineer
Wyomissing Area School District

Server Information

JSS URL: http://10.14.72.168:8080

Username: api_user

Password: jamf1234

https://github.com/mikem011/tech-talk-live-2019

Introduction to the Jamf Pro API

What does API stand for?

- Application Programming Interface
- An intermediate software layer that allows an application to talk to another application.

What is REST API?

- REpresentational State Transfer
- A format for transferring representations of objects between a client and server.

Introduction to the Jamf Pro API

CRUD Operations

Create = POST

Read = GET

Update = PUT

Delete = DELETE

Introduction to the Jamf Pro API

Classic API vs Jamf Pro API

Documentation available at <server address>:8443/api	Documentation available at <server address>:8443/uapi/doc
Reads XML or JSON	Reads JSON only
Writes XML only	Writes JSON only
Basic Authentication	Token Authentication

*The Jamf Pro API is actively being developed and will eventually replace the Classic API.

Introduction to the Jamf Pro API

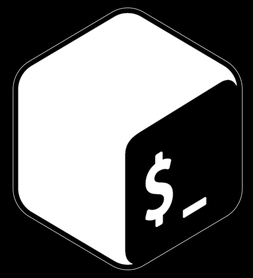
XML vs JSON

- If no content type is specified, Classic API will return XML by default
- The Jamf Pro API will ONLY return JSON
- No built in support for parsing JSON in macOS
- Python does have a built in module for parsing JSON

Introduction to the Jamf Pro API

Creating a Service Account

- Service account should not have more privileges than neccessary
- Create a read-only service account for scripts that only retrieve objects
- Create a different service account for scripts that modify objects
- Deny access to Jamf Pro Server Settings
- Deny access to Jamf Pro Server Actions
- Deny access to all Jamf Pro applications
- Use long, difficult passwords that can must be copied/pasted



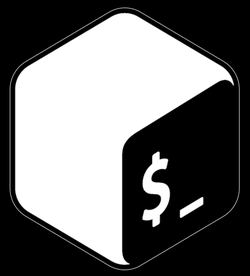
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
<jss url>:<port>/JSSResource/categories
```

Result

```
<?xml version="1.0" encoding="UTF-8"?><categories><size>5</
size><category><id>4</id><name>iOS Apps</name></
category><category><id>3</id><name>macOS Apps</name></category>…
```



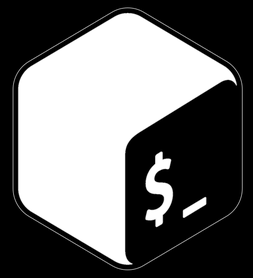
Working with the Bash Shell

Command

```
echo $(/usr/bin/curl --user "api_user":"jamf1234" \
<jss url>:<port>/JSSResource/categories) | xmllint --format -
```

Result

```
?xml version="1.0" encoding="UTF-8"?>
<categories>
  <size>5</size>
  <category>
    <id>4</id>
    <name>iOS Apps</name>
    ....
```



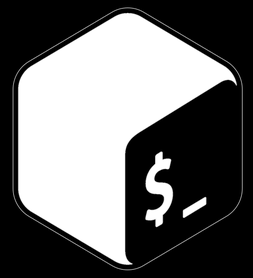
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
--header "Accept: text/xml" \
<jss url>:<port>/JSSResource/categories
```

Result

<?xml version="1.0" encoding="UTF-8"?><categories><size>5</size><category><id>4</id><name>iOS Apps</name></category><category><id>3</id><name>macOS Apps</name></category>…



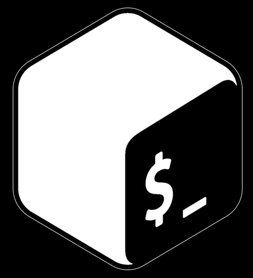
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
--header "Accept: application/json" \
<jss url>:<port>/JSSResource/categories
```

Result

{"categories":[{"id":4,"name":"iOS Apps"},{"id":3,"name":"macOS Apps"},{"id":1,"name":"Packages"},{"id":2,"name":"Scripts"},{"id":5,"name":"tvOS Apps"}]}



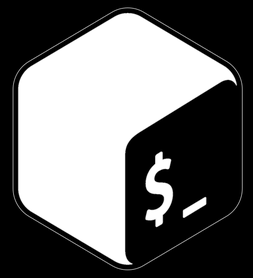
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
--header "Content-Type: text/xml" \
--request POST \
--data "<category><name>tvvOS Apps</name></category>" \
<jss url>:<port>/JSSResource/categories
```

Result

```
<?xml version="1.0" encoding="UTF-8"?><category><id>5</id><</
category>
```



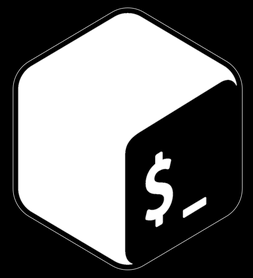
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
--header "Content-Type: text/xml" \
--request PUT \
--data "<category><name>tvOS Apps</name></category>" \
<jss url>:<port>/JSSResource/categories/id/5
```

Result

```
<?xml version="1.0" encoding="UTF-8"?><category><id>5</id><</
category>
```



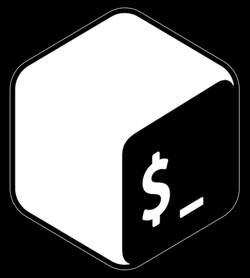
Working with the Bash Shell

Command

```
/usr/bin/curl --user "api_user":"jamf1234" \
--request DELETE \
<jss url>:<port>/JSSResource/categories/id/5
```

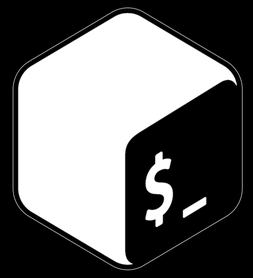
Result

```
<?xml version="1.0" encoding="UTF-8"?><category><id>5</id><</
category>
```



Troubleshooting with the Bash Shell

200	Request successful
201	Request to create or update object
400	Bad request
401	Authentication failed
403	Invalid permissions
404	Object not found
409	Conflict
501	Internal server error



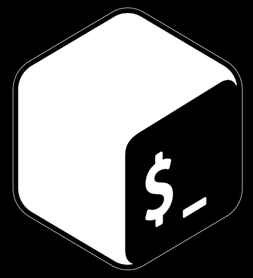
Troubleshooting with the Bash Shell

Command

```
result=$(/usr/bin/curl  --write-out "%{http_code}" \
--user "api_user":"jamf1234" \
<jss url>:<port>/JSSResource/categories)

echo ${result: -3}
```

Result

200



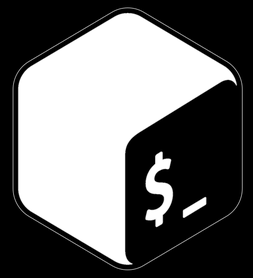
Bash Shell Examples

Retrieve Computer Count

```
/usr/bin/curl --user "api_user":"jamf1234" \
<jss url>:<port>/JSSResource/computers | sed -n -e 's/.*<size>\'
```

Retrieve Mobile Device Count

```
/usr/bin/curl --user "api_user":"jamf1234" \
<jss url>:<port>/JSSResource/mobiledevices | \
sed -n -e 's/.*<size>\(.*\)<\/size>.*/\1/p'
```



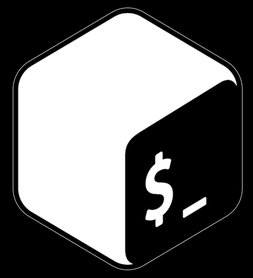
Bash Shell Examples

Delete Computer

```
/usr/bin/curl --user "api_user":"jamf1234" \
--request DELETE
<jss url>:<port>/JSSResource/computers/id/<computer id>
```

Delete Mobile Device

```
/usr/bin/curl --user "api_user":"jamf1234" \
--request DELETE
<jss url>:<port>/JSSResource/mobiledevices/id/<mobile device id>
```



Bash Shell Examples

Delete All Classes

for n in {<min_id>..<max_id>}; do \
/usr/bin/curl --user "api_user":"jamf1234" \
--request DELETE \
<jss url>:<port>/JSSResource/classes/id/$n; done

Working with Python Scripts

python>_

Working with Python Scripts

```
#!/usr/bin/env python

import sys, urllib2, base64
import xml.etree.ElementTree as ET

def main():
  request = urllib2.Request('<jss url>:<port>/JSSResource/computers/id/0')
  request.add_header('Authorization', 'Basic ' + base64.b64encode('api_user:jamf1234'))
  request.add_header('Content-Type', 'text/xml')

  try:
      response = urllib2.urlopen(request)
  except urllib2.URLError, e:
      print(e)

if __name__ == '__main__':
      sys.exit(main())
```

Working with Python Scripts

```
...
try:
    response = urllib2.urlopen(request)
except urllib2.URLError, e:
    print(e)

device_xml = ET.fromstring(response.read())
print ET.tostring(device_xml)
```

Working with Python Scripts

```
<mobile_device>
  <general>
    <id>26</id>
    <display_name>1250</display_name>
    <device_name>1250</device_name>
    <name>1250</name>
    <asset_tag>07400</asset_tag>
    <os_version>11.4.1</os_version>
...
```

Working with Python Scripts

```
...
try:
    response = urllib2.urlopen(request)
except urllib2.URLError, e:
    print(e)

device_xml = ET.fromstring(response.read())
print device_xml.find('general/name').text
```

Working with Python Scripts

```
try:
    response = urllib2.urlopen(request)
except urllib2.URLError, e:
    print(e)


device_xml = ET.fromstring(response.read())
groups = device_xml.findall('mobile_device_groups/mobile_device_group')
for group in groups:
    print group.find('name').text
```

Working with Python Scripts

```
…
request.get_method = lambda: 'POST'

POST_xml = '''<mobile_device><general><name>Test Device 1</
name><wifi_mac_address>ab:cd:ef:12:34:56</
wifi_mac_address><serial_number>SN123456789</serial_number><asset_tag>00001</
asset_tag></general></mobile_device>'''
try:
    response = urllib2.urlopen(request, ET.tostring(POST_xml))
except urllib2.URLError, e:
    print(e)
```

Working with Python Scripts

```
...
request.get_method = lambda: 'POST'

POST_xml = ET.Element('mobile_device')
general_elem = ET.SubElement(POST_xml, 'general')
ET.SubElement(general_elem, 'name').text = 'Test Device 1')
ET.SubElement(general_elem, 'wifi_mac_address').text = 'ab:cd:ef:12:34:56')
ET.SubElement(general_elem, 'serial_number').text = 'SN123456789')
ET.SubElement(general_elem, 'asset_tag').text = '00001')

try:
    response = urllib2.urlopen(request, ET.tostring(POST_xml))
except urllib2.URLError, e:
    print(e)
```

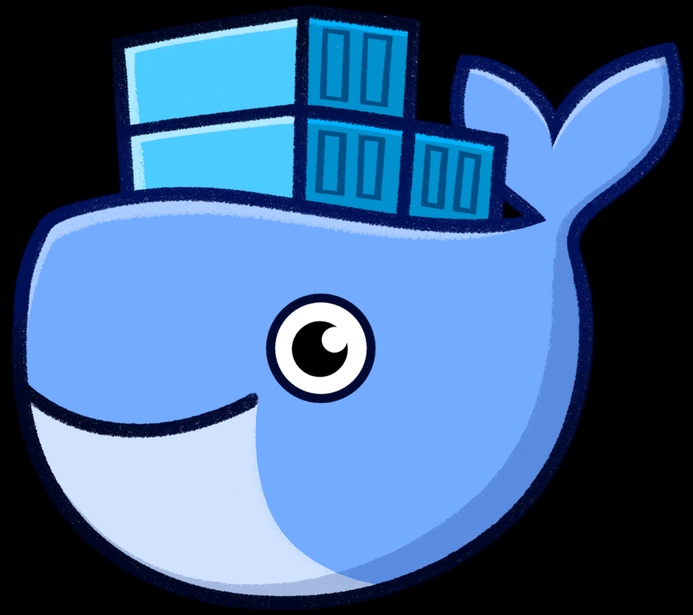
Working with Python Scripts

```
request = urllib2.Request('<jss url>:<port>/JSSResource/mobiledevices/id/51')
…
request.get_method = lambda: 'PUT'

PUT_xml = ET.Element('mobile_device')
general_elem = ET.SubElement(PUT_xml, 'general')
ET.SubElement(general_elem, 'name').text = 'Test Device 2')
location_elem = ET.SubElement(PUT_xml, 'general')
ET.SubElement(location_elem, 'username').text = 'bharper')

try:
    response = urllib2.urlopen(request, ET.tostring(PUT_xml))
except urllib2.URLError, e:
    print(e)
```

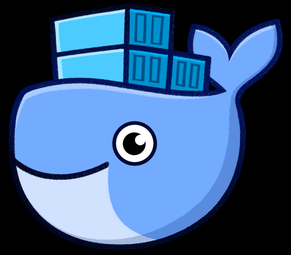
Working with Python Scripts

```
request = urllib2.Request('<jss url>:<port>/JSSResource/mobiledevices/id/51')
…
request.get_method = lambda: 'DELETE'

try:
    response = urllib2.urlopen(request)
except urllib2.URLError, e:
    print(e)
```

Docker




I DONT ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION



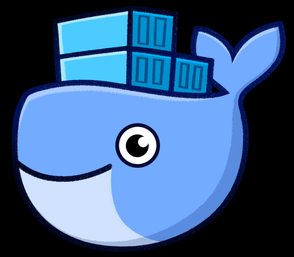
Docker & Docker-Compose

Installation

Download the Community Edition from Docker Hub:
https://hub.docker.com/search/?type=edition&offering=community

Docker Components

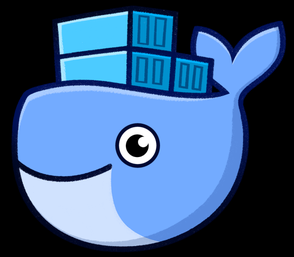
- Docker Engine
- Docker-Compose
- ~~Docker Swarm~~



Docker & Docker-Compose

Docker Commands

docker pull <image name>
docker images
docker run <image name>
docker start <container id>
docker stop <container id>
docker rm <container id>
docker ps -a
docker logs <container id> -f
docker inspect <container id>
docker exec -it <container id> /bin/bash

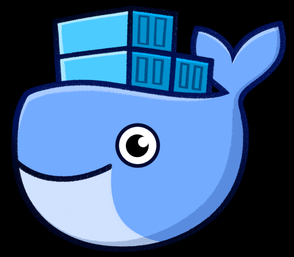


Docker & Docker-Compose

Docker-Compose Commands

docker-compose config

docker-compose up -d



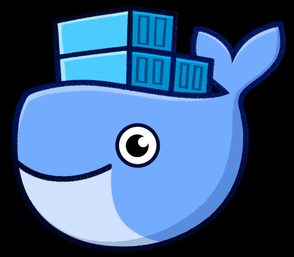
Docker & Docker-Compose

Download the Images

docker pull mysql
docker pull jamfdevops/jamfpro:0.0.7

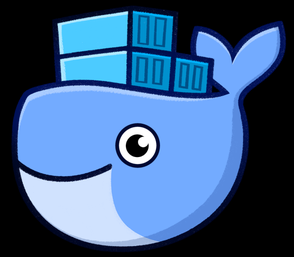
Run the Images

docker run -v /path/to/local/mysql/storage:/var/lib/mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD='mysecretpassword' -e MYSQL_USER='jamfsoftware' -e MYSQL_PASSWORD='jamfsw03' -e MYSQL_DATABASE='jamfsoftware' -d --name jss_mysql --restart mysql; docker run -v /path/to/local/jss/storage:/data/ROOT.war -v /path/to/local/tomcat/storage:/usr/local/tomcat/webapps -p 8080:8080 -e DATABASE_HOST='jss_mysql' -e DATABASE_NAME='jamfsoftware' -e DATABASE_USERNAME='jamfsoftware' -e DATABASE_PASSWORD='jamfsw03' -e DATABASE_PORT='3306' -d --name jss --restart jamfdevops/jamfpro:0.0.7



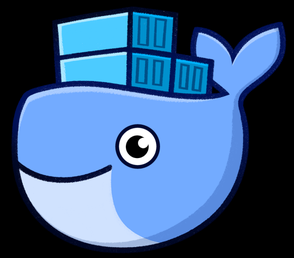
Docker & Docker-Compose

docker-compose



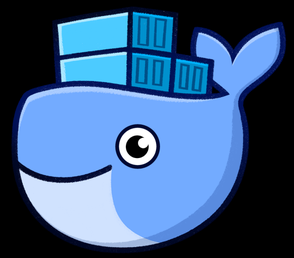
Docker & Docker-Compose

```
docker-jamf
|__docker-compose.yml
|__jss.env
|__mysql_volume
|__mysql.env
|__ROOT
|  |__ROOT.war
|__webapps
```



Docker & Docker-Compose

```yaml
version: '3.7'
services:
  mysql:
    image: 'mysql:5.7'
    container_name: jss_mysql
    volumes:
      - '/Users/mike/Desktop/docker-jamf/mysql_volume:/var/lib/mysql'
    restart: 'always'
    ports:
      - '3306:3306'
    env_file:
      - ./mysql.env
  jss:
      depends_on:
        - mysql
      image: 'jamfdevops/jamfpro:0.0.7'
      container_name: jss
      volumes:
        - type: bind
          source: /Users/mike/Desktop/docker-jamf/ROOT/ROOT.war
          target: /data/ROOT.war
          consistency: cached
        - type: bind
          source: /Users/mike/Desktop/docker-jamf/webapps
          target: /usr/local/tomcat/webapps
          consistency: cached
      ports:
        - '8080:8080'
      env_file:
        - ./jss.env
```



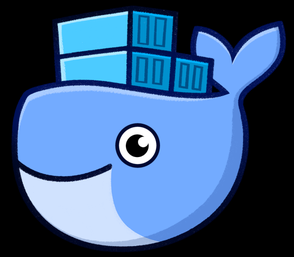
Docker & Docker-Compose

MySQL ENV File

```
MYSQL_ROOT_PASSWORD=mysecretpassword
MYSQL_USER=jamfsoftware
MYSQL_PASSWORD=jamfsw03
MYSQL_DATABASE=jamfsoftware

```

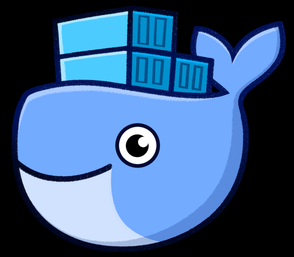
JSS ENV File

```
DATABASE_HOST=jss_mysql
DATABASE_NAME=jamfsoftware
DATABASE_USERNAME=jamfsoftware
DATABASE_PASSWORD=jamfsw03
DATABASE_PORT=3306

```



Docker & Docker-Compose

docker-compose up -d



Docker & Docker-Compose

Jamf-Migrator

Developing an App with Swift

Apps & Resources

- Postman
- The MUT
- JSS_Helper
- Spruce
- Jamf Pro Groups Scoped
- Jamf Marketplace - https://marketplace.jamf.com
- Jamf Developer Site - https://developer.jamf.com

Questions?