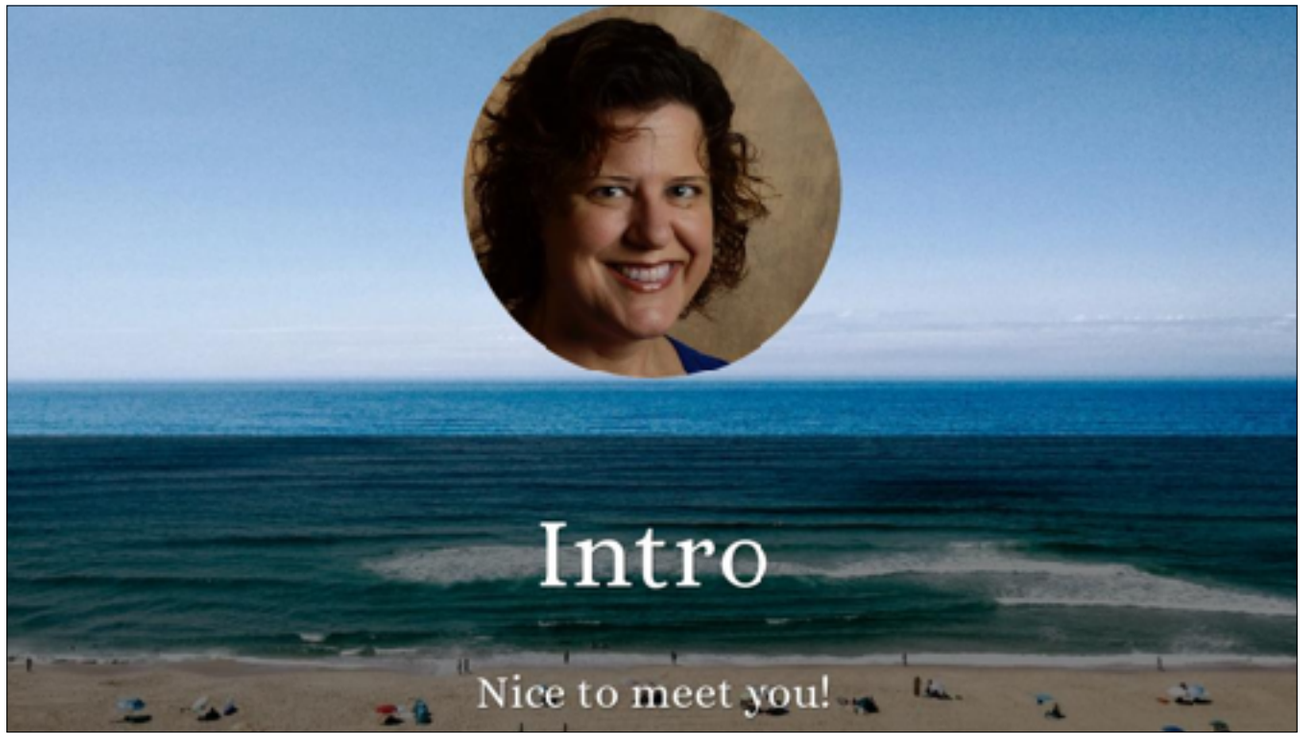
Return of the CLI



API Evangelist: Akamai
Worked with APIs for over 10 years
Frustrated with un-usable APIs
Excited by CLIs

* Evolution of UI over time
*Understand advantages of each approach

90’s: Command line tools to perform various tasks

- Relatively friendly to techies
- Unusable by end users
* Help was challenging
- Chain commands

- Web portals became a much more popular UI over time
  - Enabled use by users who were neither developers nor sysadmins
  - Widgets and helpers
  - Portal itself: cannot be automated
  - Automation: Integration with low level APIs

- APIs allowed external developers to create tools to interact with the platform, usually alongside a web portal
  - Requires code to create tools
  - Very customizable
  - Can be automated
  - Not very friendly for sysadmins

- CLIs evolved
  - Targeted to specific use cases OR API wrappers
  - No code required
  - Automatable
  - Friendly to both developers and sysadmins
* Not friendly to end users
  - Perfect for DevOps

Can handle your authentication and other environment settings for you

* Create/Spin up instances
  - Configure
  - Control

Google, Microsoft, Amazon… Akamai

System Administration

Operations and automation

Uses in system administration

- Automation
- Sequencing tasks
  * Configuring systems
  * Integrating systems together

- Time costs
- Aggregation
- Common use cases
* API Wrapper
- Pain points in UI and API
- Authentication

Successful API -> Many users
Difficult interface -> time and frustration for *ALL*
Time spent on CLI -> multiplied by customers using it

increases loyalty
In your best interests
focus API dev efforts

Common

* Lack of automation
* User input
* UI limitations
* API Challenges

Akamai:

* Creating new properties
* Activating properties
* Viewing large lists of items
* Adding or removing hostnames

… business logic/code

Some focus on API Wrappers
I'm talking more about use case based CLIs
Ideally an API is use case driven, but…

What are your customers doing?
Where are they getting stuck?
How can you improve their experience?

Akamai:

- Want to create properties
- Update/modify specific pieces
- Activate and deactivate
- Delete

API has tens of endpoints
Focused on just a few

On the other side are CLIs that wrap the API interface

Authentication
Friendly naming
Automatable

User friendly -> to people comfortable with a command line

CLI Examples

From across the industry

Examples of CLIs out in the wild.
Some are API wrappers
Some are use-case based tools.

Docker is a well known container technology
It's like a small virtual machine with very little overhead
No need to tune memory/size
Create containers which can be reused
Used by some development teams to enforce consistency

Surprising: It is a CLI/API system
The CLI is a blend of wrapper and use case based tasks

Let's take a look at a couple of examples.
API interaction first and CLI version second

API Interaction

```
$ curl --unix-socket /var/run/docker.sock -H "Content-Type: application/json" \
  -d '{"Image": "alpine", "Cmd": ["echo", "hello world"]}' \
  -X POST http:/v1.24/containers/create
{"Id":"1c6594faf5","Warnings":null}

$ curl --unix-socket /var/run/docker.sock -X POST http:/v1.24/containers/1c6594faf5/start

$ curl --unix-socket /var/run/docker.sock -X POST http:/v1.24/containers/1c6594faf5/wait
{"StatusCode":0}

$ curl --unix-socket /var/run/docker.sock "http:/v1.24/containers/1c6594faf5/logs?stdout=1"
hello world
```

Most docker interaction is done via the CLI, but there is a backend API.
"docker run" is a great example of a use-case based CLI command
- First create the container
- Then start it
- Wait until it completes
- Show the logs

In this case we're using the API
- Read the documentation
- Set curl flags appropriately
- Run the commands
- No client to handle errors
- No command line help

CLI Interaction

I'd show the output of docker run —help but it's pretty long.
The number of options and parameters is astounding

I think this example speaks for itself.
The CLI is the most friendly way to interact with docker
Still it has some excellent shortcuts

UX/DX should be a driver for a CLI as it is here

Try the newer native docker client. Check out what people are doing with it.

Well designed CLI for the purpose.

Heroku

Heroku is a Platform as a Service company

Create/test your code on your system
Push it into the cloud

They have a CLI, an API and a Dashboard
1/3 of our users use the CLI only
2/3 use the CLI and dashboard
a very small amount only use the dashboard only or the backend APIs

CLI covers almost all API functionality
Designed to be the main interface

Makes sense that they would use the CLI for this functionality
Most people are working in the command line already
Provide a usable CLI based on workflows

Good documentation and tutorials.
Give it a try to see what it feels like.

Amazon is a huge purveyer of APIs.
Everything has an API
Most of their systems are built API/CLI
Internal mandate for feature parity
CLI is not designed for use cases
Does handle authentication and other configuration

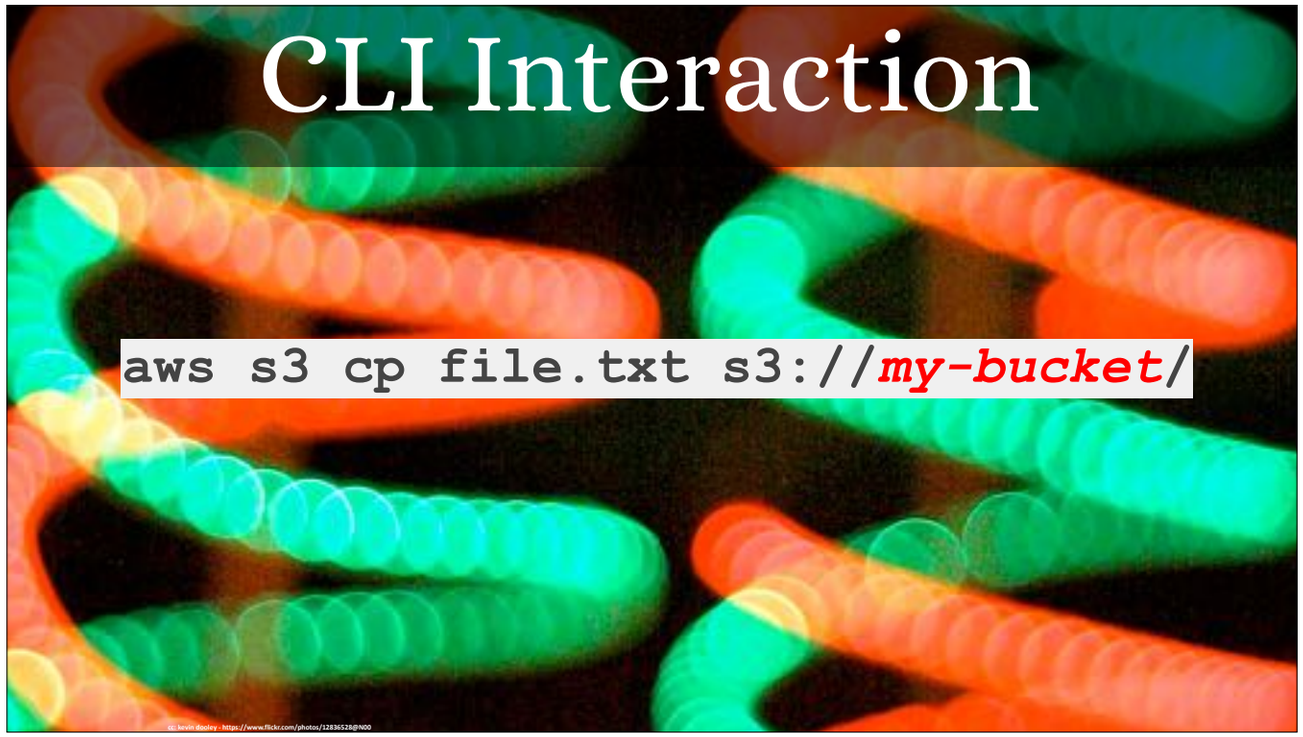
They have zillions of APIs
Let's take a look at S3

API Interaction

CLI calls don't wrap multiple API calls
But API calls are complicated

Won't go over all the pieces here
The API requires an inordinate amount of work

Can you use it? Sure.
Would you want to? Probably not without an interaction library.



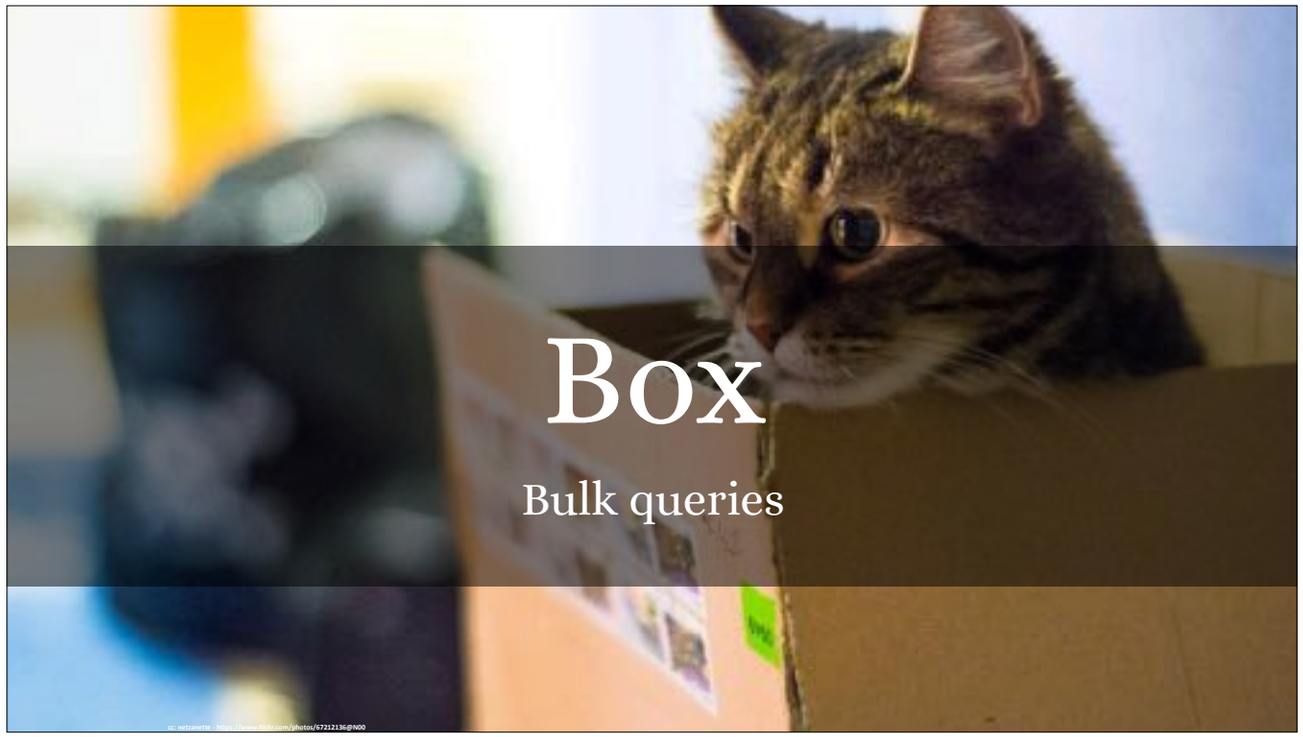
Here, I'll make this one easy.

That's a user interface I can understand.

Note:

- Most interaction with AWS needs Business Logic
- Many things can be done with the console
- Documentation can be frustrating

You can use the CLI to do a great number of things in AWS.
Spin up new EC2 instances.
Launch a Lambda function.
Pretty much anything in AWS.
Sometimes the console is easier though.



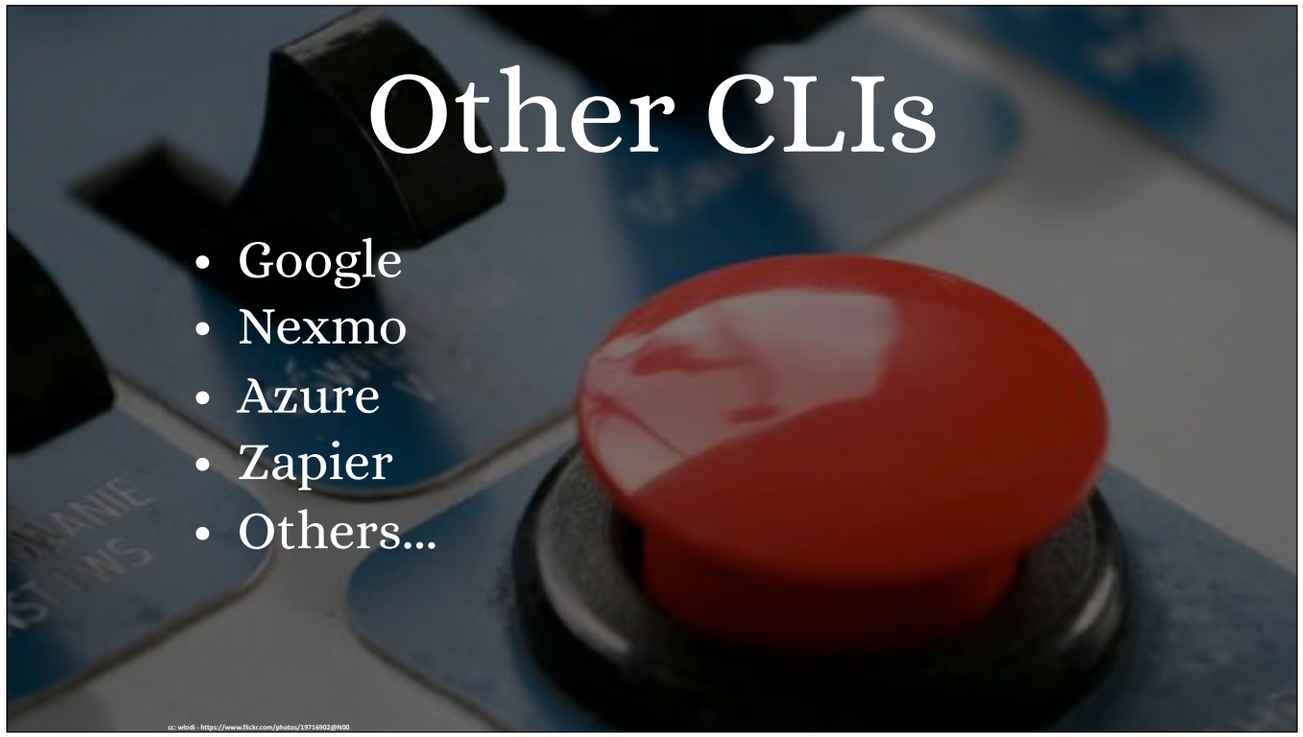
Box just last week came out with their CLI

The CLI largely echoes the functionality of the API, but it allows bulk processing.

Being able to mass-delete or update files on the system enables automation.

Most people will use the dashboard.

Hooking the CLI into your publication process.



I cannot possibly list all of the companies with CLIs out there.
Most of them are the "big boys"
Smaller companies are adding this interaction model
I personally prefer platforms with both an API and CLI

Many products have a strong dashboard
Great for one-time actions
Easy to understand
Widgets and helpers

but

It's not automatable
Can't incorporate it into CI/CD
It can't be an integrated part of your process

Luna, Akamai Web Portal
Big improvement over the PS world

ACCESSIBLE TO EVERYONE

No command line experience

Anyone can use it
Widgets and helpers
Visual context
Obviously, no code

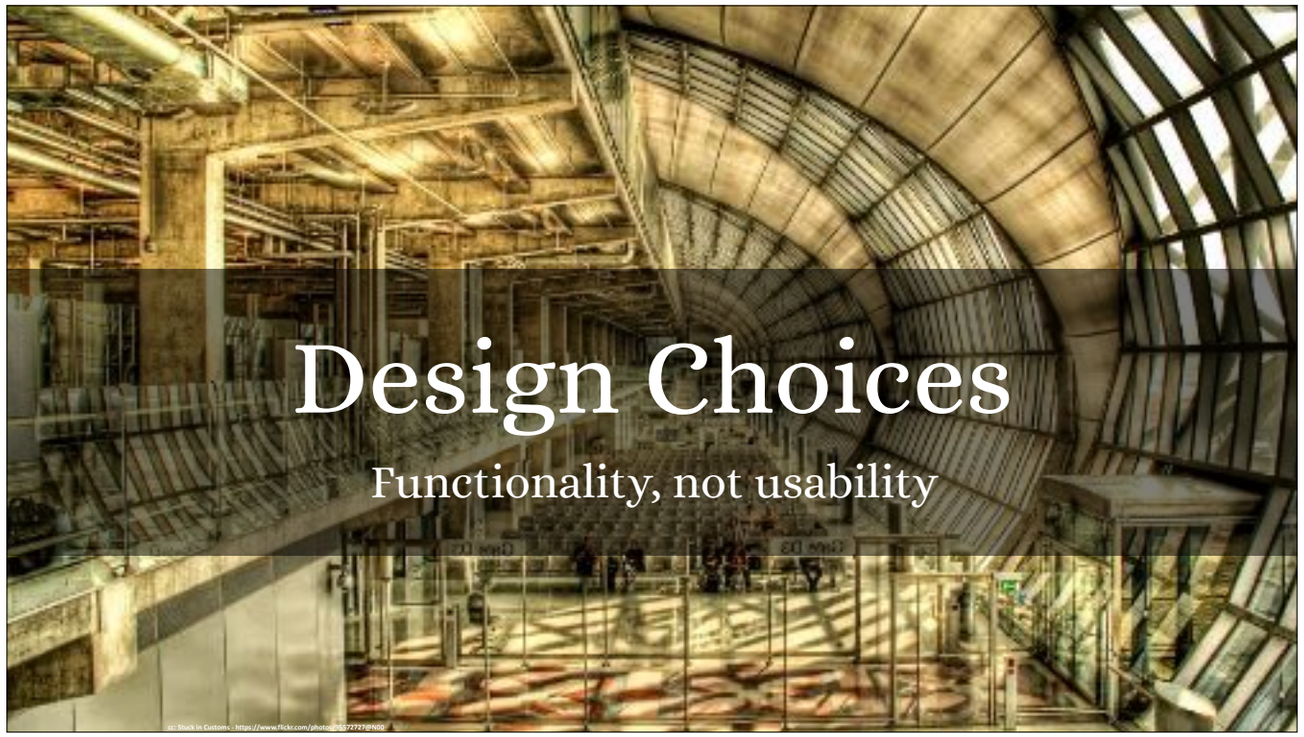
FULL FUNCTIONALITY

Create, edit, activate, delete

Luna enables all property functions
Property management plus account products
Reporting available

Portal maintains the user's context
Understands the business logic
Navigation through properties
Configuration of features

Property Manager API (PAPI)
Created to allow customers programmatic access to property functions
Fully functional, can do everything PM can do



PAPI was designed for full functionality
Very strong REST methodology
Individual items are individually addressable
Possible to do just about anything but…

The system wasn't designed for usability
Designed for functional completeness

Accounts/groups/properties/hostnames
Hierarchy is needed to work with PAPI
Internal property ID is required

In order to use the API
Must find the property (new search endpoint)
Understand group and contract
Make several calls to perform a single action

PAPI not the only function
- GTM
- Purge
- cloudlets

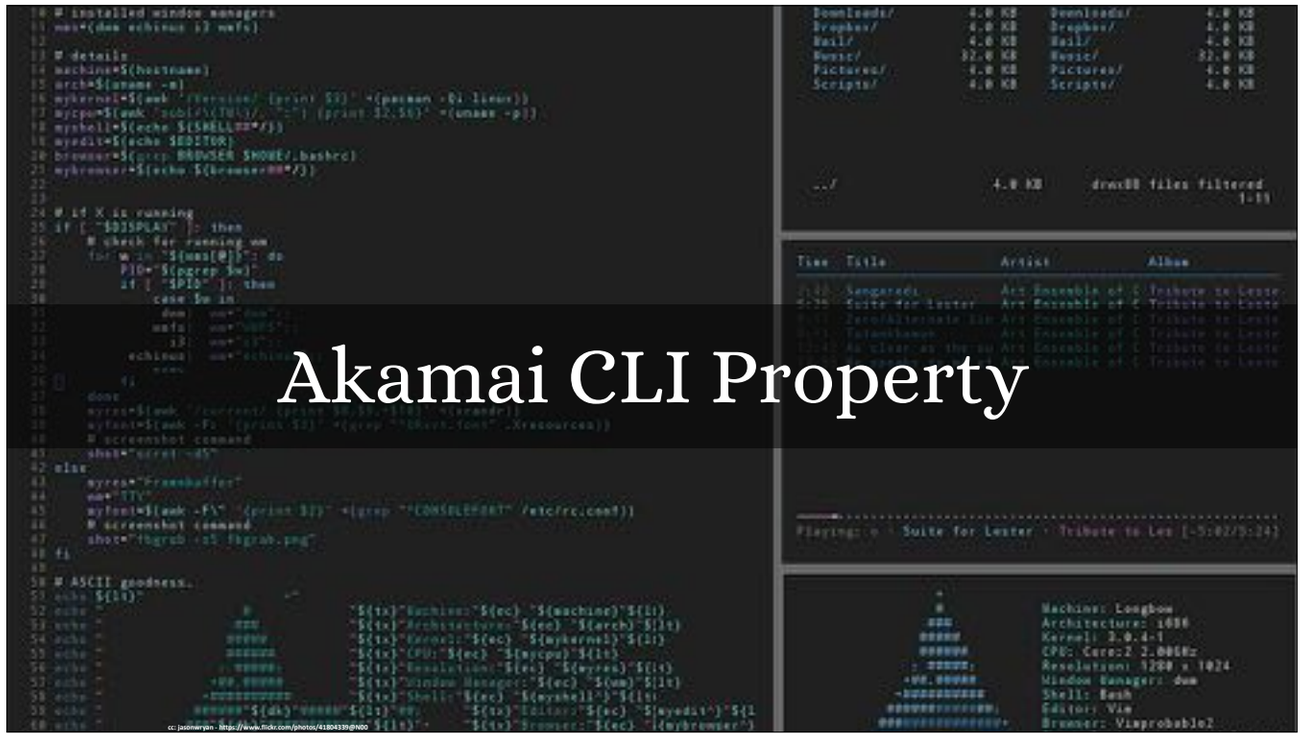
Created the Akamai CLI package manager
- Multiple commands
- Languages
- Consistent interface
- Installation, updating

Extensible

Support for multiple languages

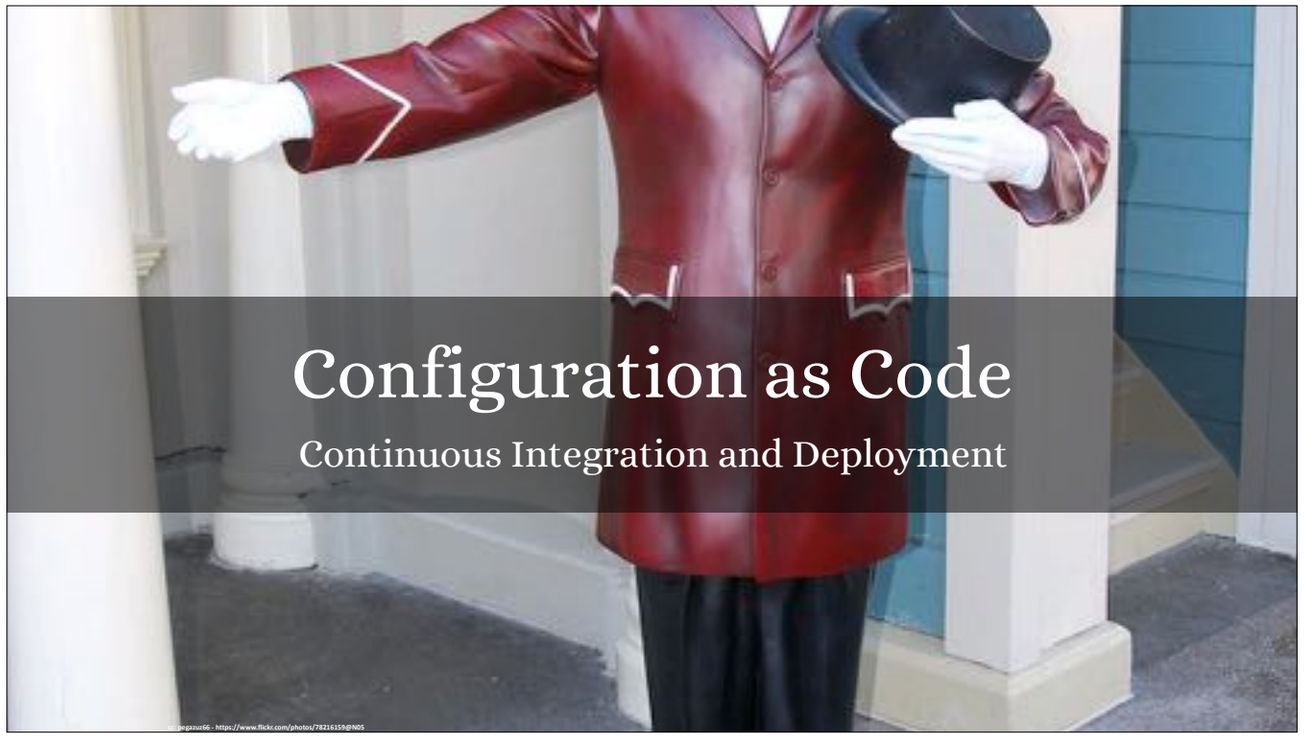
CLI system is extensible
New packages created all the time
CLI is in Go for precompiled binaries
Packages are in Node, python, and go

New development model
Packages are open source
Pull requests and issues in github



CLI property - devops focus
Entire life cycle of a property
Abstracts business logic
Can be chained together

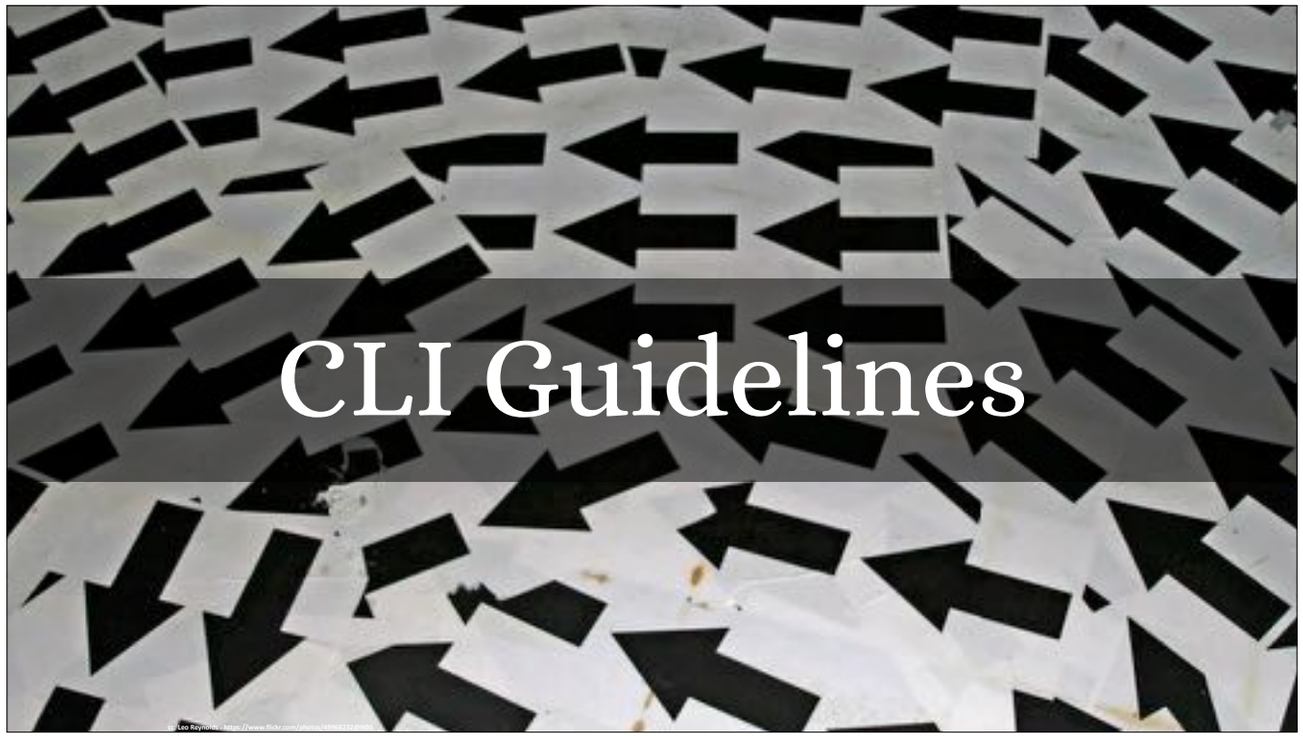
Create
Update
Activate/Deactivate
Delete



Jenkins example

- Configuration in an SCM
- Commits trigger build
- Create temporary property
- Update with new rules
- Activate property
  - <perform your tests>
- Delete property

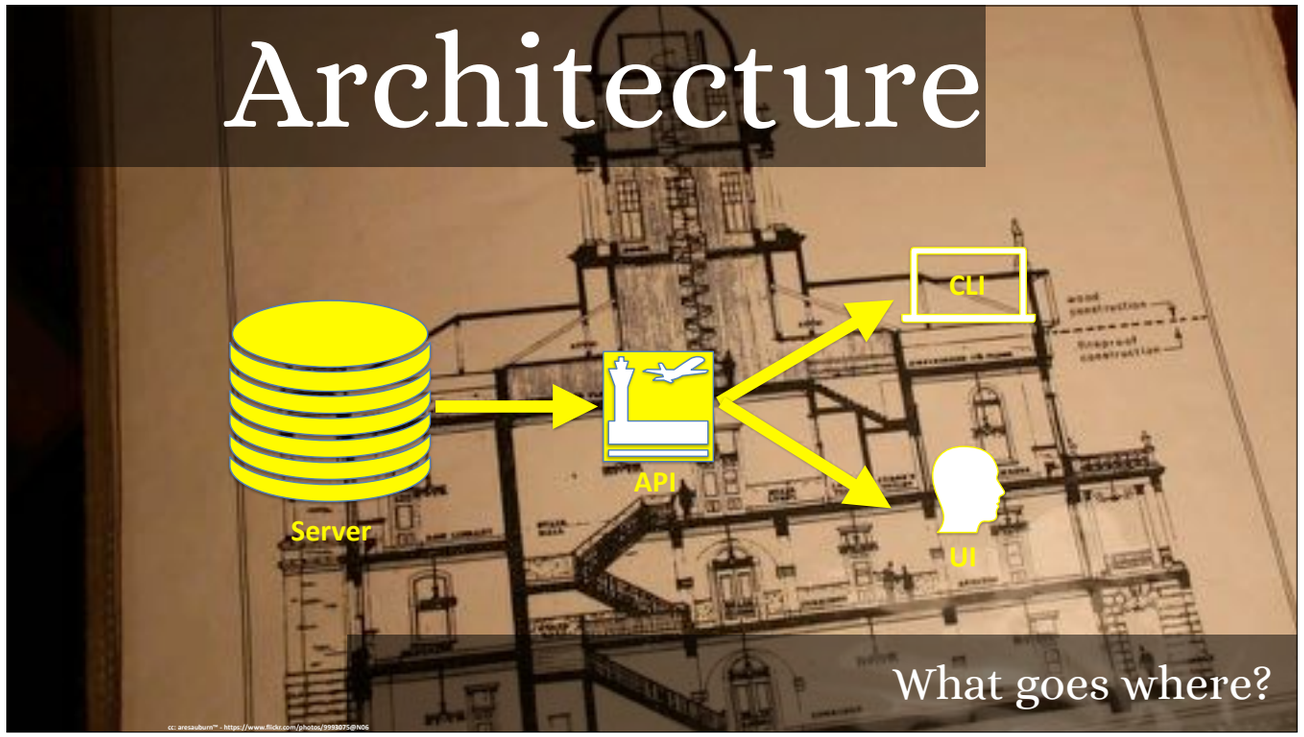
CLI commands are done with property name
<shoes.departmentstore.com>
No need for hierarchy
Works when properties move



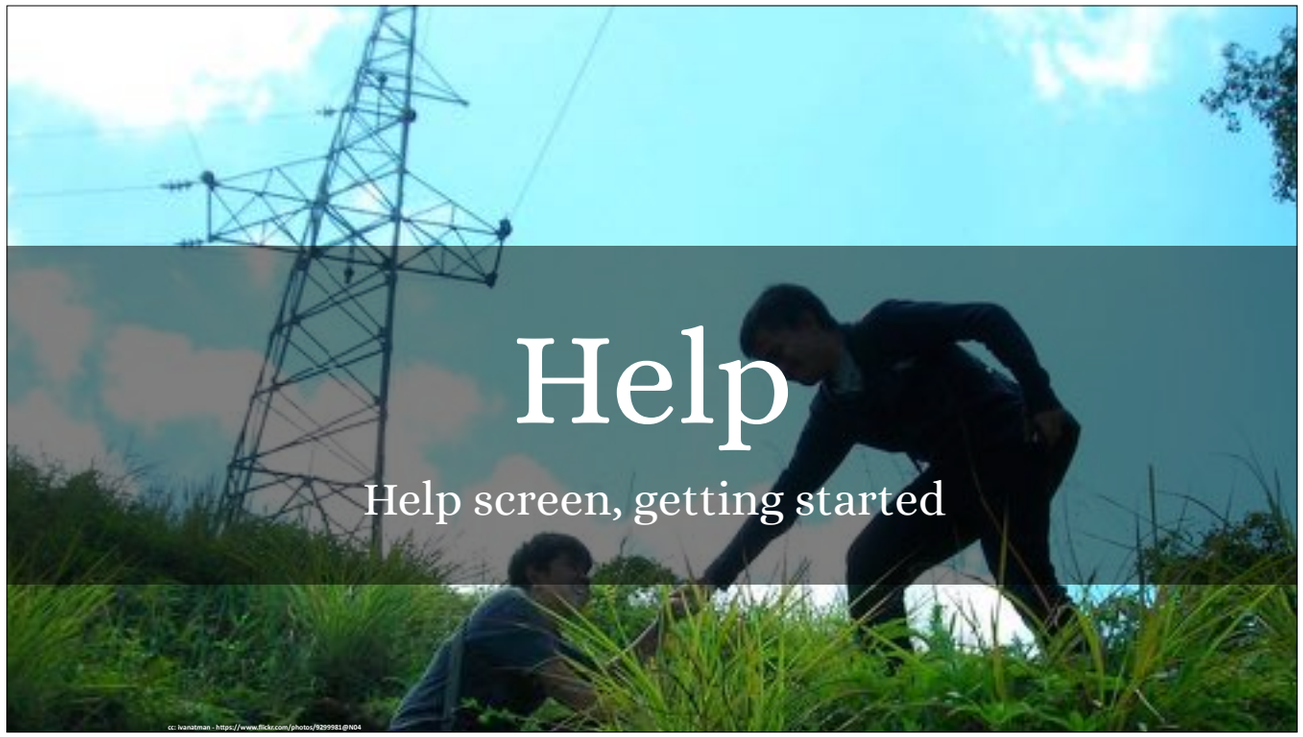
Guidelines for CLI
Check your mirrors

Consider API-CLI-UI parity
- Important for wrappers
- Use case based maybe no

Architecture



How to structure
Server in the middle
Chain where UI calls CLI
Best: API in front of all clients



Humans using CLI
Consistent interface
Help screens
Examples and tutorials
First class product

cc: Tau Zero - https://www.flickr.com/photos/744418@N08

OS Support

Meet your customers where they are

Created many tools
Linux/Mac support
Customers use Windows
Wrapper supports linux, mac, windows
Test on all platforms

Study git and docker
Command/subcommand/flags/target
Look at common unix commands
Don't reinvent the wheel

Advantage to CLI is environment
Easy authentication
Configuring behavior
Allow consistency for multiple runs

Soapbox

- APIs should be usable
- Most of them are not
- Fixing them is hard

CLI for API wrapping is useful for environment

Consider case based CLI tools

- Find customers pain points with UI *and* API
- Create interfaces that ease those issues
- Understand customers' use cases (like devops)
- Allow customers to choose between CLI and API