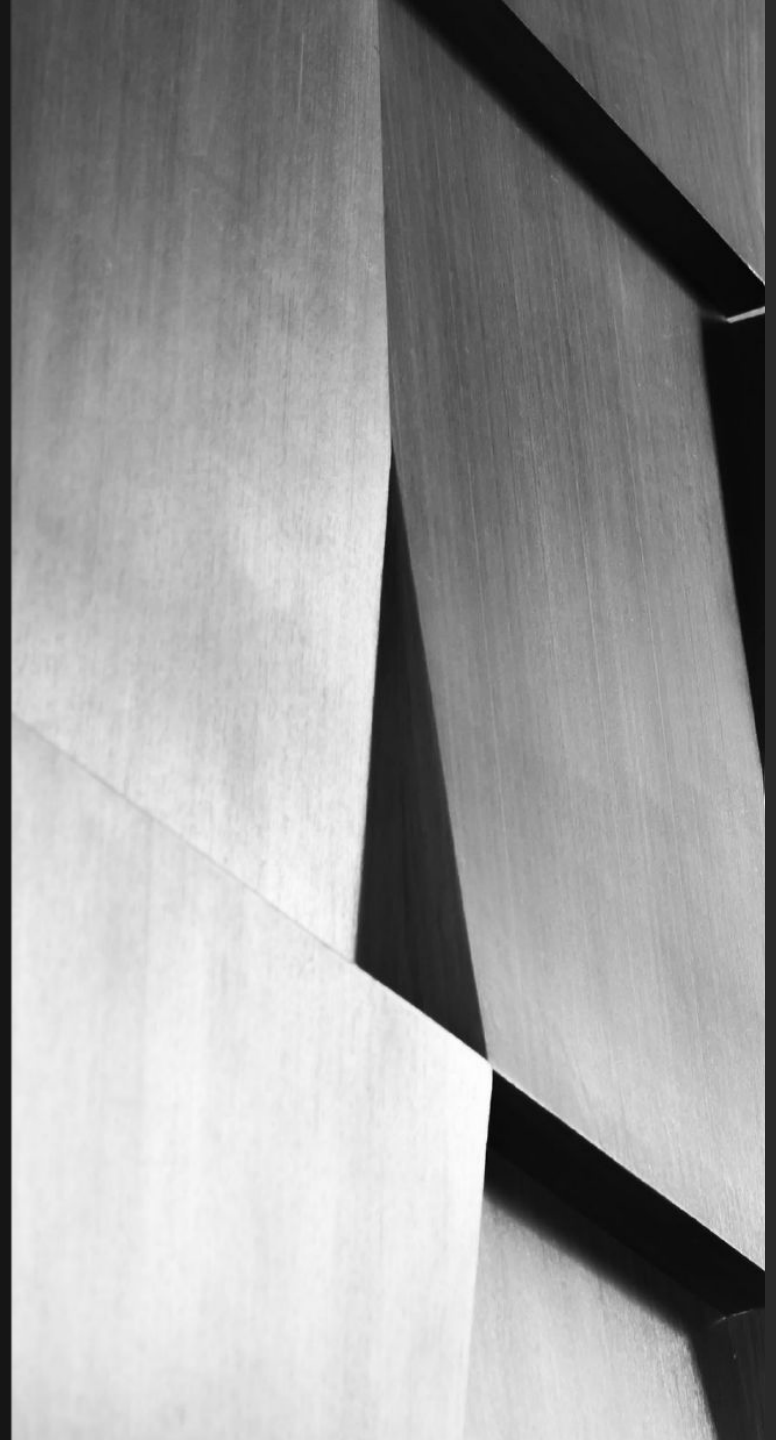
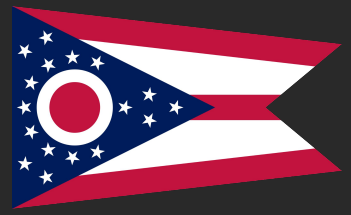


Embracing Patterns

Automation by Design

Brad Krumme
Automation Evangelist
Senior Specialist Solution Architect - Automation @ Red Hat

A Bit About Me:



Native Ohioan

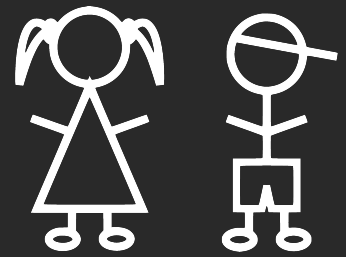
Over 20 Years in Tech

NERD!



Enjoyer of Whiskey

Father to 2 Humans







Slave to 2 Felines

Let’s start with Patterns…

A natural or chance arrangement of component parts, traits, or elements

Building an Animal using Parts/Traits/Elements from Nature



Slender & Muscular Body



Strong & Flexible Forelimbs



Sharp & Curved Claws

Let’s put all of those traits together…



- Slender & Muscular Body
- Strong & Flexible Forelimbs
- Sharp & Curved Claws
- **IT’S A CAT!!!!**

Now let's add some details...

- It should weigh 300 pounds
- It needs to have round pupils
- The fur should be orange with black stripes
- **IT'S STILL A CAT!!!**



~~A natural or chance~~

An arrangement of component parts, traits, or elements

Building an App using Parts/Traits/Elements in Technology





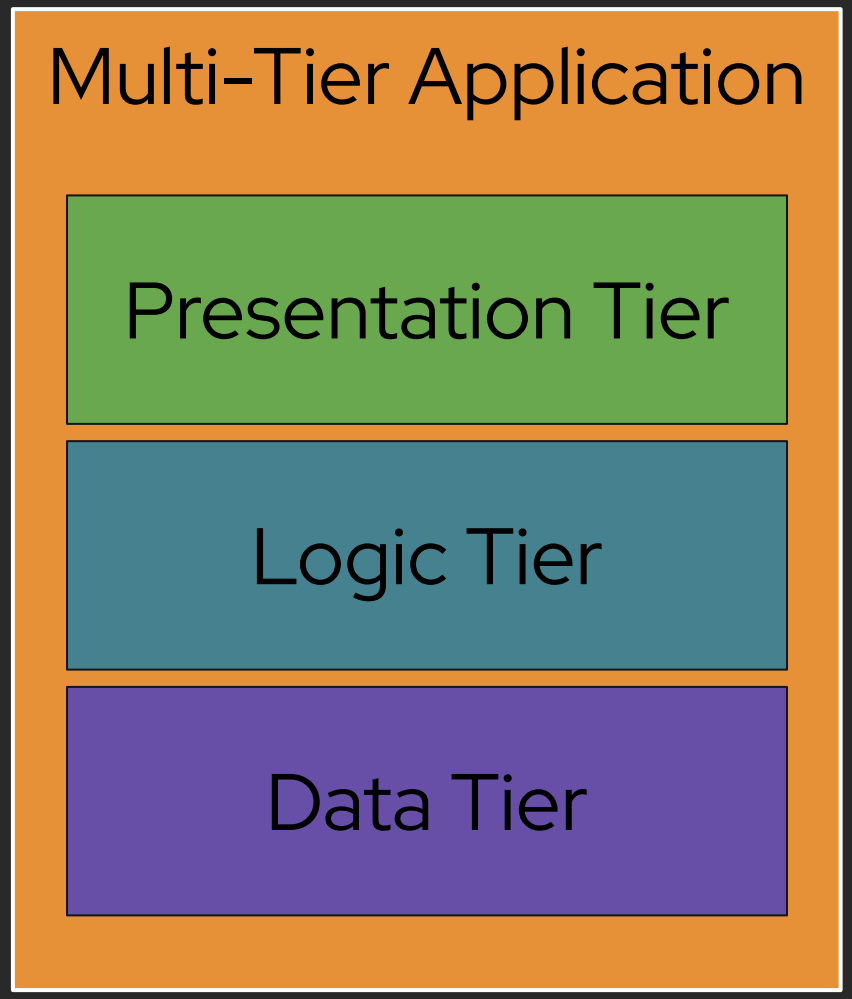


Web-Based
Frontend
Interface

Powerful
API-Based
Backend

Extensible
Relational
Database

Let's put all of those features together...

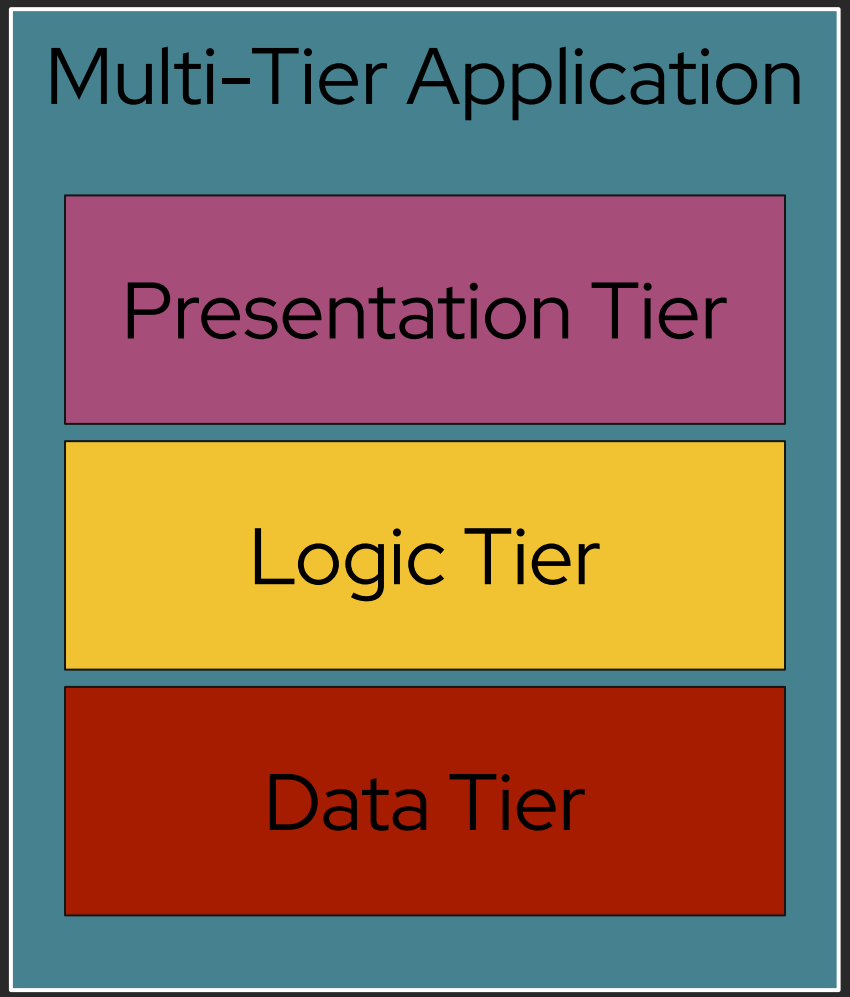


- Web-based frontend interface
- Powerful Api-Based Backend
- Extensible Relational Database

Now let's add some details...

It's still a >>>

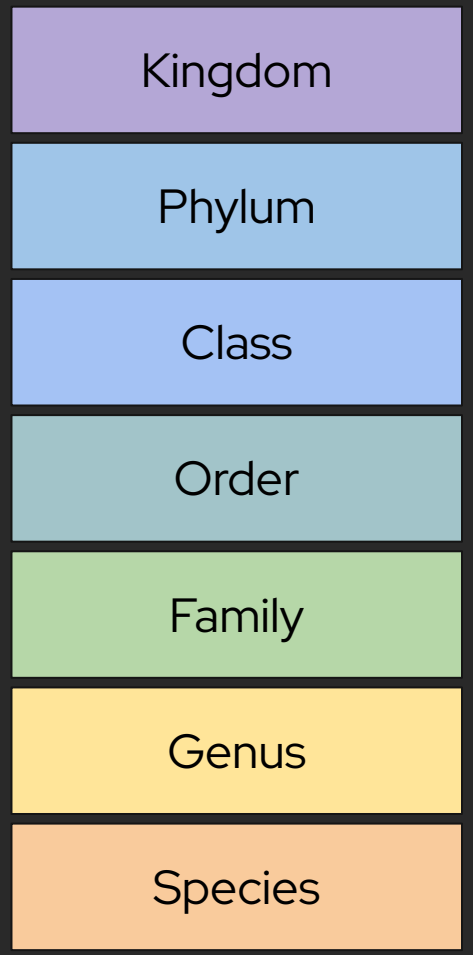
- The presentation tier is HTML
- The logic tier code is .NET
- The data tier is SQL Server



How should we view our IT services and applications?

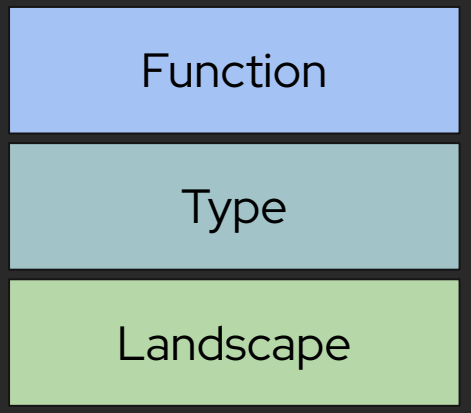
Biological Taxonomy

- Many layers
- Groups organisms based on shared characteristics
- Uses physiological, molecular, behavioral, and ecological details
- Lowest step on the ladder is a specific species

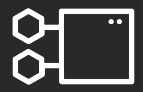


Technological Taxonomy

- Fewer layers
- Groups components based on function
- Subdivides functions using types
- Lowest step on the ladder is a specific application stack



More Details about the Technological Taxonomy



Gateway



Frontend



Backend



Database

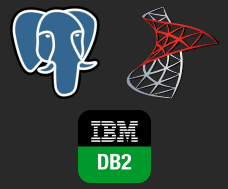
Function

Describes the high-level ***purpose or role*** of a layer or component of an application or service









Type

Describes the ***composition or variety*** of a component type



Landscape

The ***specific implementation*** for an aggregate set of functions and their chosen types

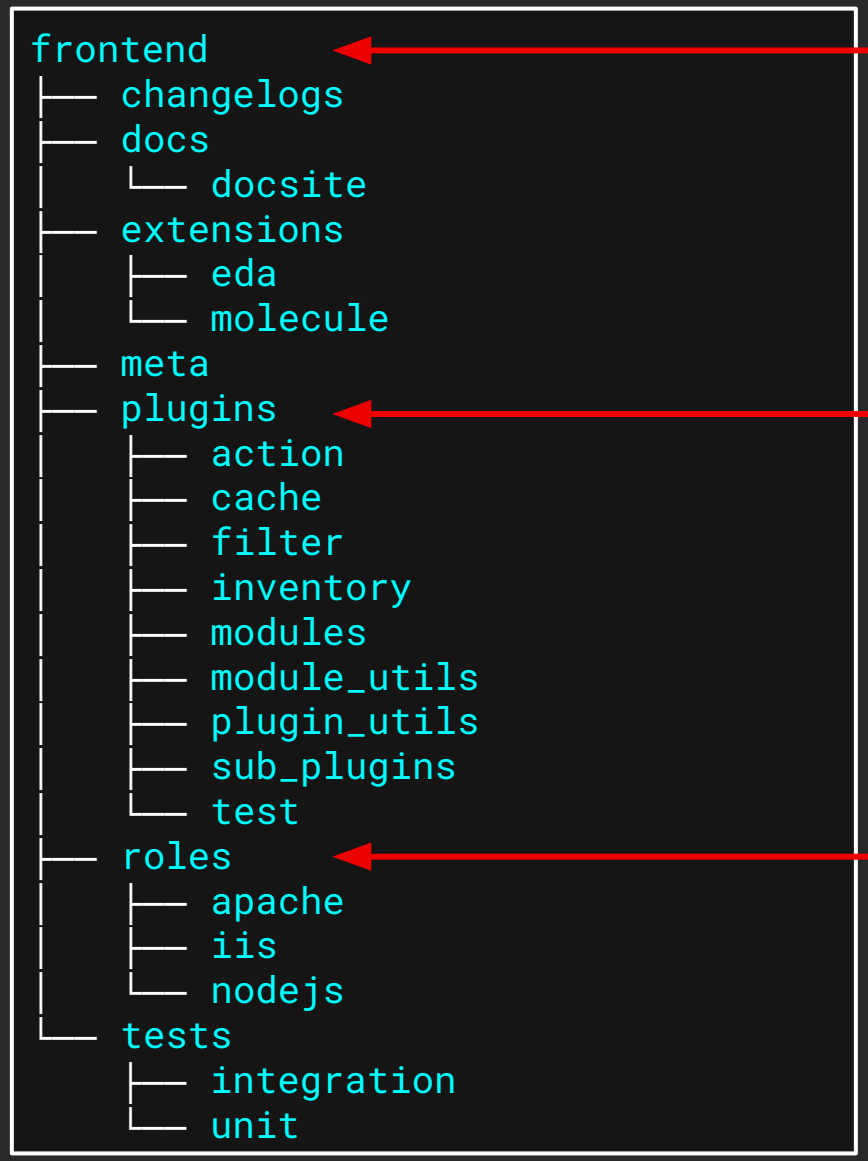
“OK, but I thought this talk was about automation…”

- *Everyone in the room*

The Technological Taxonomy with **Ansible**

- **Functions** are *building blocks of* application stacks or *Landscapes*
  - Represented as Ansible ***Collections***
- **Types** are *consistent implementations* of specific technologies to deliver a *Function*
  - Represented as Ansible ***Roles***
- **Landscapes** are *combinations* of Functional *Types* to build an *application stack*
  - Represented as Ansible ***Plays and Playbooks***

Ansible Collection Project Example



Collection name matches ***Function***

Each Functional component of an application architecture has its own Ansible Collection

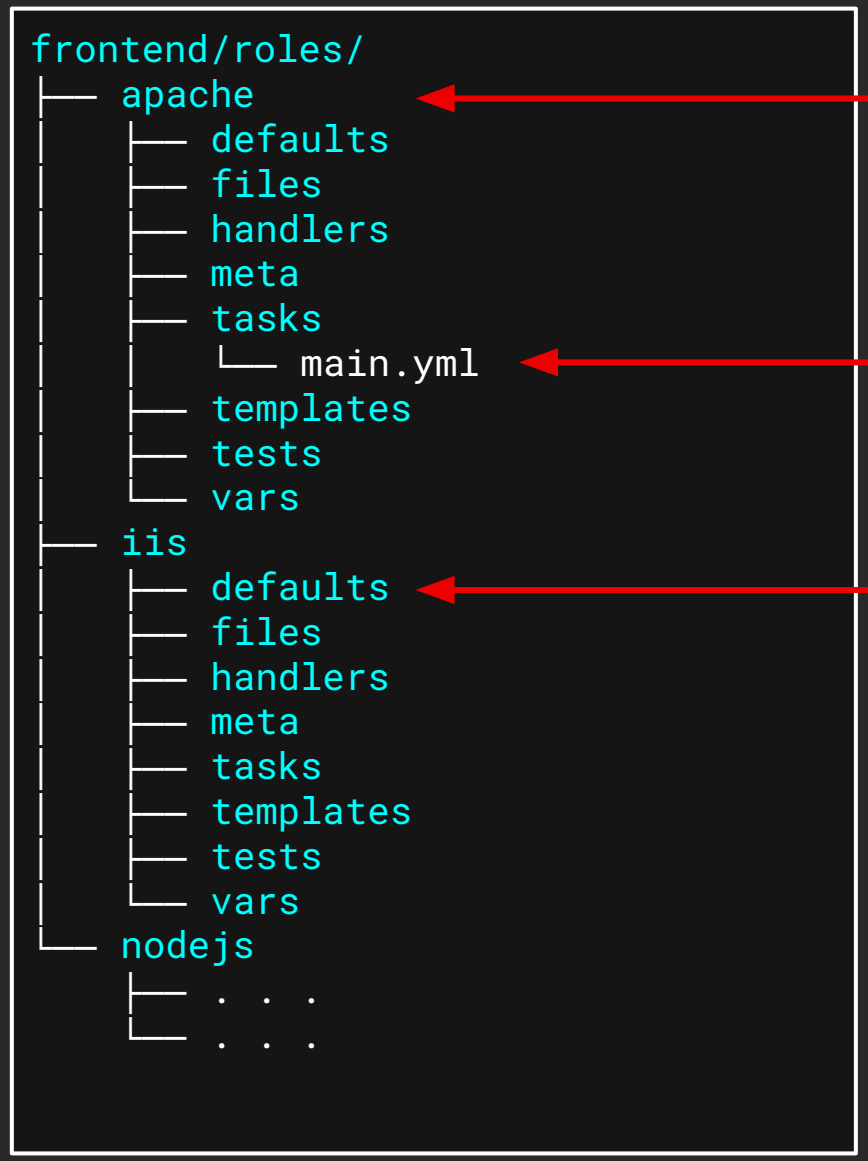
Collection might include ***Ansible Plugins*** or ***Modules***

For use cases which don't have modules or plugins available from the Ansible community (or Ansible Automation Hub for Red Hat customers)

Collection includes ***Ansible Roles*** for each ***Type***

Each collection has one or more roles for the different types of the function

Ansible Role Structure Example



Role name matches ***Type***

Each type will have its own Ansible Role used for configuring the details of the application component

Role functionality is modularized

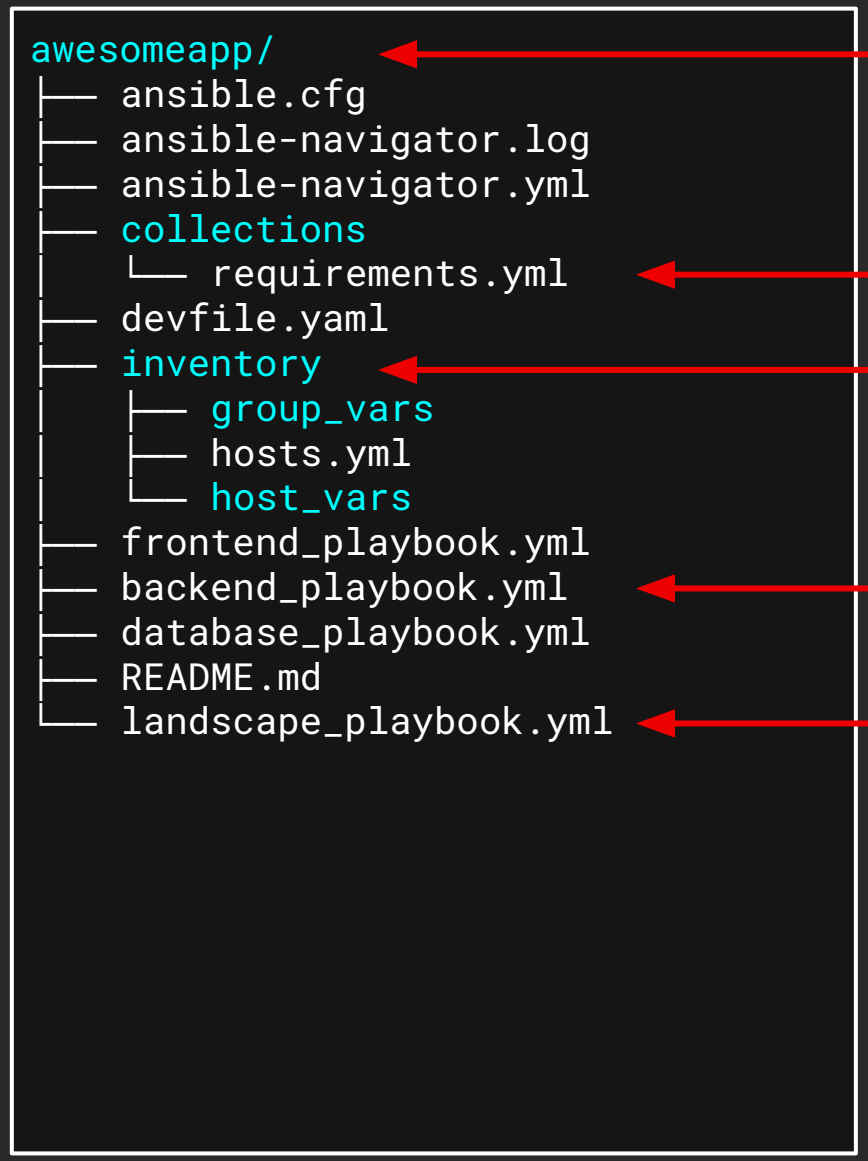
Standard Ansible Role structure enables reuse across multiple applications and teams

Roles have pre configured default variables

Each role should include default variables which are tested and known to provide a basic instance of the type

Separation of Automation Function and Configuration is Imperative

Ansible Playbook Project Example



Project name matches ***Landscape***

Collection requirements are defined in the project

Inventory is a directory structure

*The inventory includes the **To-Be** configuration defined as variables in **group_vars** and **host_vars***

Playbooks for each Function of the Landscape

A consolidated ***landscape_playbook.yml*** playbook

A "playbook of playbooks" which can be used to configure the entire stack in a single motion
This is sometimes named site.yml

Let's organize our site-specific data

- Identify your **Single Source(s) of Truth** and use them for Ansible Inventory
  - Data from multiple SSOTs should not overlap or contradict each other
  - Combine SSOTs into one inventory using ***dynamic inventory plugins***
- Differentiate between **As-Is** and **To-Be** information
  - Discovered ***Ansible Facts*** from the existing environment is the **As-Is**
  - Managed **To-Be** information is represented as ***Variables*** in your ***Ansible Inventory***
- Define your ***Ansible Inventory*** as a structured directory instead of a single file
- Restrict the usage of variable types
  - Opt for ***Inventory Variables*** instead of play variables, include_vars, and vars_files
  - Avoid Scoped Variables and Extra Variables

Ansible Inventory File Example

```
$ cat inventory/hosts.yml
---
all:
  hosts:
    webserver1:
    webserver2:
    appserver1:
    appserver2:
    dbserver1:
    dbserver2:
  children:
    web_servers:
      hosts:
        webserver1:
        webserver2:
    app_servers:
      hosts:
        appserver1:
        appserver2:
    db_servers:
      hosts:
        dbserver1:
        dbserver2:
```

- Each ***Function*** has its own *inventory group* in the inventory
- ***Variables*** should be set in *group_vars* and *host_vars* instead of the inventory file for simplicity of variable precedence
- ***Inventory groups*** can contain other groups
- **Individual hosts** can exist in multiple groups

Ansible Inventory Variables Examples

```
$ cat inventory/group_vars/web_servers.yml
---
apache_http_port: 80
apache_https_port: 443
```

```
$ cat inventory/group_vars/db_servers.yml
---
postgresql_listen_port: 5432
```

```
$ cat inventory/group_vars/all.yml
---
app_version: "1.0.0"
```

```
$ cat inventory/host_vars/webserver1.yml
_
ansible_host: 192.168.1.101
server_name: webserver1
```

- Each ***inventory group*** has its own *group_vars* definition file where variables are set for the ***Type***
- Variable names/keys should be *meaningful* with a prefix that matches the ***Ansible Role*** name
- Variables which apply to ***all hosts*** in the inventory are defined in *all.yml*
- **Individual hosts** can have their own specific variables in *host_vars*

Ansible Playbook Examples

```
$ cat frontend_playbook.yml
---
- name: Configure frontend webservers
  hosts: web_servers
  become: true
  roles:
    - bpkrumme.frontend.apache
```

```
$ cat backend_playbook.yml
---
- name: Configure backend appservers
  hosts: app_servers
  become: true
  roles:
    - bpkrumme.backend.jboss
```

```
$ cat database_playbook.yml
---
- name: Configure database servers
  hosts: db_servers
  become: true
  roles:
    - bpkrumme.database.postgresql
```

```
$ cat landscape_playbook.yml
---
- ansible.builtin.import_playbook: frontend_playbook.yml

- ansible.builtin.import_playbook: backend_playbook.yml

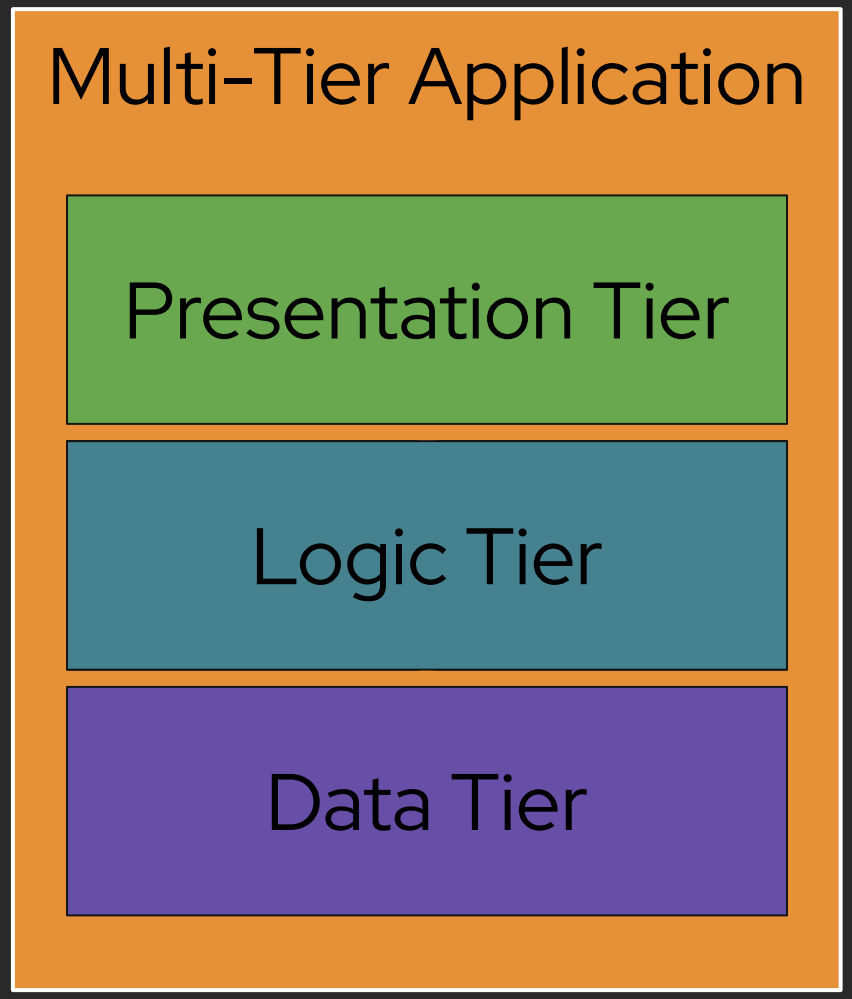
- ansible.builtin.import_playbook: database_playbook.yml
```

- ***Functions*** are contained in their own *individual playbooks* which call the ***Role*** for the individual ***Type***
- Playbooks *can include other roles* as necessary for the hosts
- The functional component *playbooks are aggregated* into a single ***Landscape*** playbook

Thanks! Now how do we scale this?

- *Everyone in the room…probably*

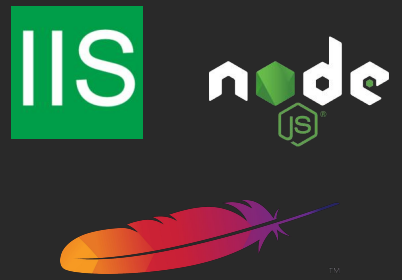
Patterns follow your architecture!



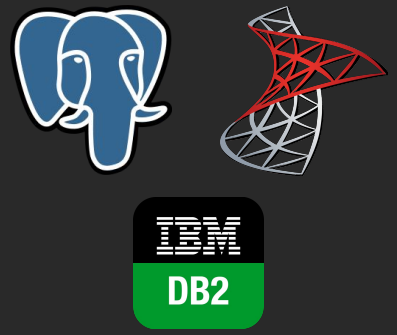
- New *architecture elements* become new ***Functions***
- New *Functions* become new ***Ansible Collections***
- New *Types* become new ***Ansible Roles***

Domain experts retain ownership of their domain

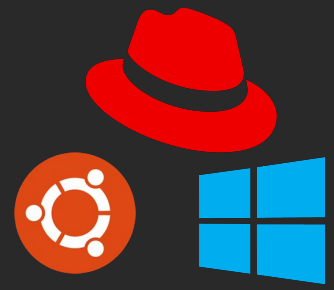
Web Servers



Database



Server OS

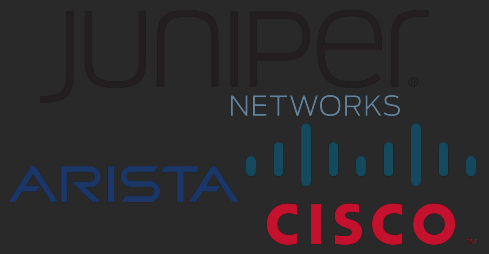


Each Team creates and maintains their own Ansible Collections





App Server



Network

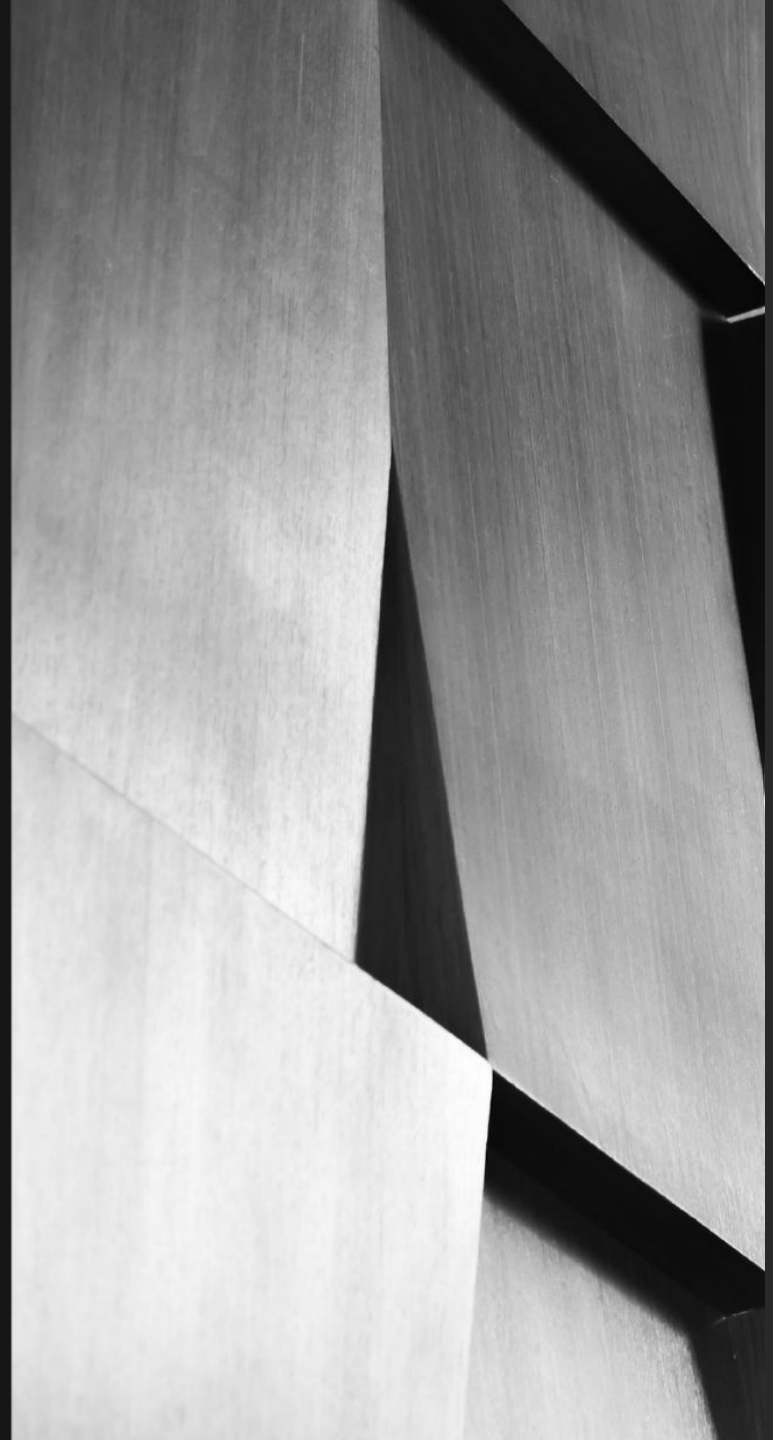


Security

Collections should be **Published, Shared, and Reused**

- ***Publish*** your Ansible Collections
  - Create release as tarball on GitHub
  - Upload to *Ansible Galaxy* if allowed/desired
  - *Red Hat AAP* includes Private Automation Hub
- ***Share*** your Ansible Collections
  - Make automation functions available to others
  - Enable standardization through democratization
- ***Reuse*** Ansible Collections as much as possible
  - Don’t reinvent the wheel!
  - Create New Collections only for new *functions*

Q&A



Thank you!!!