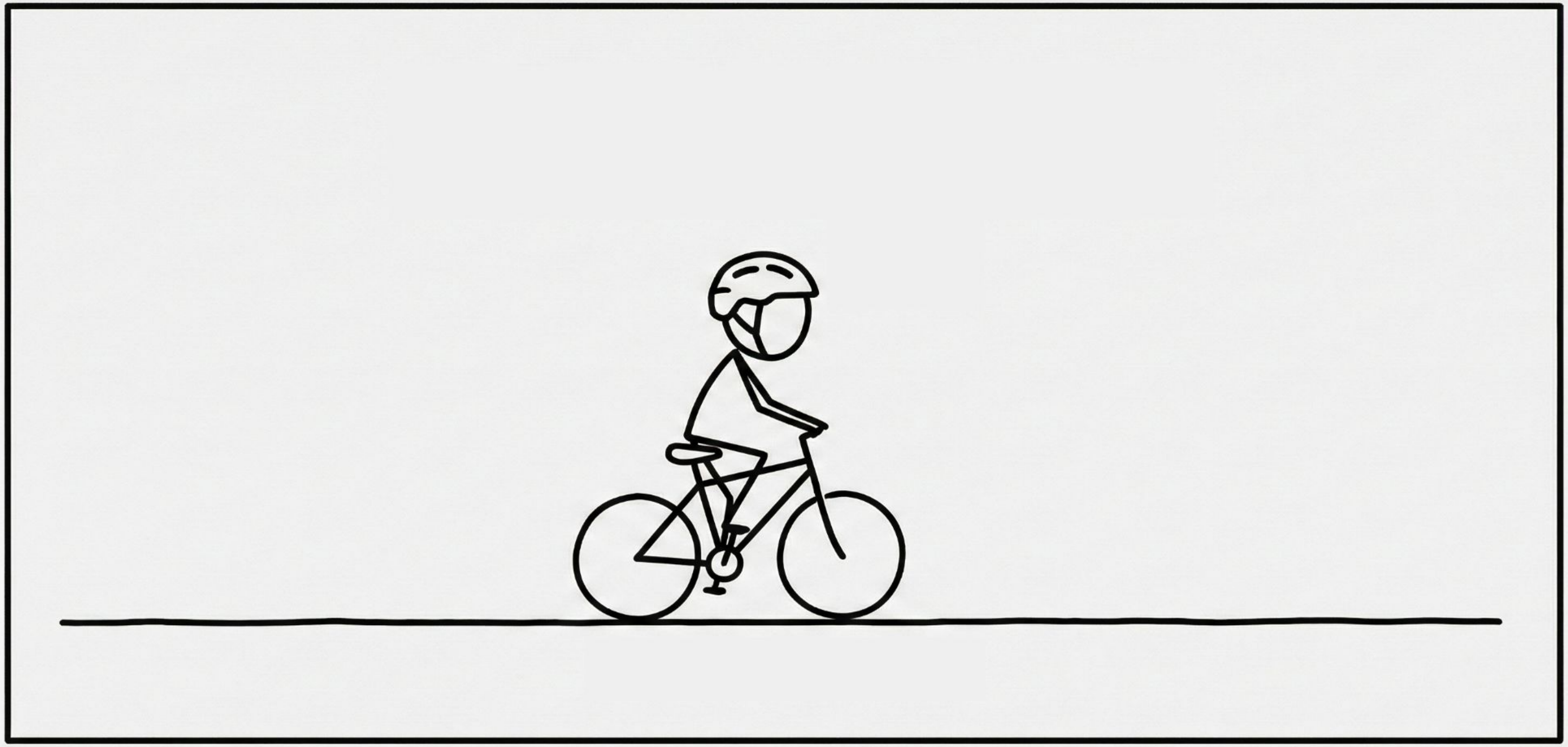
Don't Fear the Jinja

Matt Davis - Ansible Core Architect @ Red Hat
CfgMgmtCamp 2026

{{ ... }}



It looks so simple...

- It's not.
- Every {{ }} expression -> an ephemeral code-generated Python module

{{ inventory_hostname | upper }}

```
from jinja2.runtime import LoopContext, Macro, Markup, Namespace, TemplateNotFound, TemplateReference, TemplateRuntimeError, Undefined, escape, identity,
internalcode, markup_join, missing, str_join
name = None

def root(context, missing=missing, environment=environment):
    resolve = context.resolve_or_missing
    undefined = environment.undefined
    concat = environment.concat
    cond_expr_undefined = Undefined
    if 0: yield None
    l_0_inventory_hostname = resolve('inventory_hostname')
    try:
        t_1 = environment.filters['upper']
    except KeyError:
        @internalcode
        def t_1(*unused):
            raise TemplateRuntimeError("No filter named 'upper' found.")
    pass
    yield environment.finalize(context, t_1((undefined(name='inventory_hostname') if l_0_inventory_hostname is missing else l_0_inventory_hostname)))

blocks = {}
debug_info = '1=18'
```

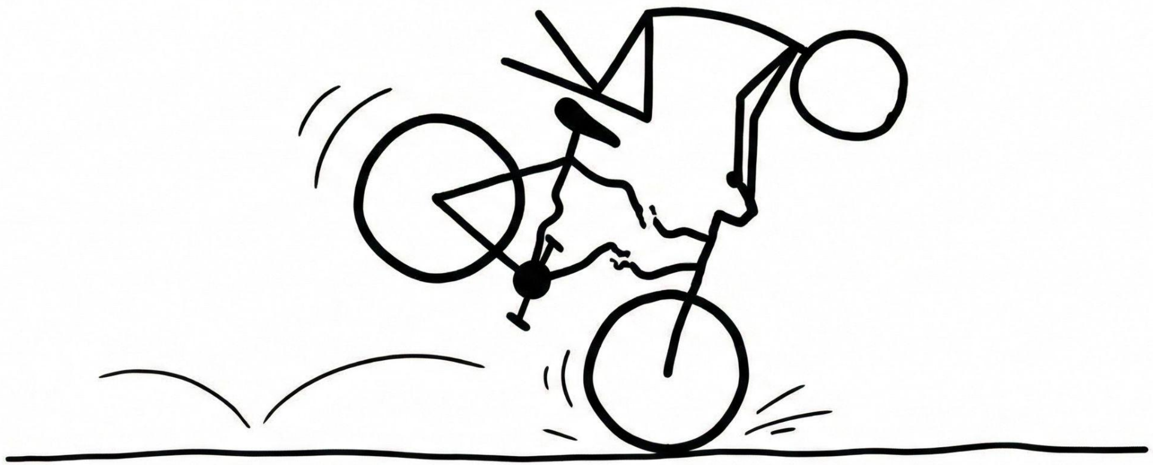
Why do we need all this?

- Technically, we probably don't.

```
- hosts: web
  tasks:
  - package:
      name:
      - apache2
      - nginx
- hosts: db
  tasks:
  - package:
      name:
      - mariadb
```

Isn't This Better?

```
- hosts: all
  tasks:
  - package:
      name: '{{ required_packages }}' 👀
```



What's Up With Quoting Handlebars?

- {{ }} default was a **very** unfortunate YAML collision
- Unquoted YAML strings **starting** with { look like inline mappings to YAML
  - {{ anywhere else in string needs no special treatment!
- Similar later systems (e.g. GHA/AZP) use ${{ }}
- Ansible *could* trivially use both {{ and ${{
  - 3-line code change
- Minor backward-compatibility problems:
  - `msg: The cost is ${{ cost }}`
  - `win_powershell: ${{target_var}} = Get-Item C:\`
  - Is $ a static part of the output or a template marker?

Jinja in Ansible - Templates

- Can contain multiple `{{ }}` expression blocks and static text
- Directive blocks `{% %}` for looping, locals, macros, includes, etc.
- Supported for text templates, variables and most playbook keywords
- The full power of Jinja

Jinja in Ansible - Expressions

- Render a single value
- Can use Jinja filter/test plugins
- Expressions are what's valid inside `{{ }}`
    - Cannot have embedded {{ }} or use {% directive blocks
- `foo | selectattr("bar", "==", "baz)`

Jinja in Ansible - Conditionals

- Ansible-specific construct
- Special expression that must yield true/false
    - `when: foo is integer`
    - `until: bar > 5`

    -
      ```
      assert:
        that: baz == 42
      ```
- Non true/false conditional result **is an error** in 2.19+

So is a Playbook a Jinja Template?

- Common misconception, but nope
- Template-ability implemented (or not) by each keyword
- Special sauce over Jinja for Ansible features
  - Automatic lazy template-on-fetch
  - Native Mode (contributed to Jinja upstream)
- Most templating occurs at `task` keyword level
  - So each playbook keyword value can be a full **template**
- Fully-dynamic play/block/task is not supported

Jinja Template Behavior in Ansible

- **Only** run on the Ansible controller
- Should not have side effects
    - May be evaluated more often than you think
    - Use custom filter/test plugins and {{ lookup("pipe" ... }} with care
- Not just strings and scalars
- Lazy recursive templating
    - Templates are not evaluated until referenced
    - Laziness was vastly improved in 2.19

DEMO - Complex Types and Laziness

Fleshing out the package sample

- Add a separate list of packages installed on all hosts?

```
- package:
    name: |-
      {{ (shared_packages | default([])) +
         (host_packages | default([])) | unique }}
```

- Starting to get a lot of logic in here- what if we need to reuse it?

Reusable Logic Options : Copy/Paste

- Simplest, but rarely the best.

Reusable Logic Options : Roles

- Lots of tasks on single hosts? Roles are OK.
- Otherwise overkill.

Reusable Logic Options : Ansible Jinja Plugins

- Custom filters can transform values
  - `foo | my_custom_filter`
  - `list_of_foo | map('my_custom_filter')`
- Custom tests can answer yes/no questions
  - `foo is my_custom_test`
  - `list_of_foo | select('my_custom_test')`
- Somewhat heavyweight - must be written in Python
- Requires packaging/distribution
- Hides complexity (+/-), good performance

Reusable Logic Options : Jinja Macros

- Easier than custom (Python) Jinja plugins
- More limited than custom Jinja plugins
- Lightweight content-embeddable inline functions (2.19+)
  - Still clunkier than it should be :(

DEMO - Var Embedded Macro

Variable Precedence and Jinja Templates

- Templates and expressions are always evaluated in a context
  - Context == "where the variable values come from"
- Each context is a many-layered cake of snapshotted variable values
- Dozens of different contexts- common ones:
  - Play (no host variables)
  - Task+Hostvars (Play + Task * Hosts)
  - Task+Hostvars+Loop ((Play + Task * Hosts) + Current Loop Values)

Variable Precedence and Jinja Templates - cont'd

- Any guesses on what this will do?

```
- name: play using {{ my_host_var }}
  hosts: all
  tasks:
  - name: task using {{ my_host_var
    shell: echo {{ my_host_var }}
```

What's Up With Templated Play/Task Names?

- Luckily a pretty isolated case. So. Many. Bugs.
  - Complex callback interactions
- Short answer: Just Don't
- If you must, use vars guaranteed to be defined above host level

Jinja Growing Pains

- No list/dict comprehensions
  - Some builtins, (e.g.: map, select) work for simple stuff
- No (supported) collection mutation
  - Ansible blocks most workarounds by default for security reasons
  - e.g. {% set _=foodict.__setitem__(k, v) %}
- Composition is harder than it should
- Performance cliffs
  - Oh the stories...
  - Please file issues for things that are notic
- Exercise:
  - Use Jinja against **`setup`** to get a list of a addresses. Not impossible, but harder th

Jinja Growing Pains - Solutions

- KEEP IT SIMPLE
  - Could/should you do this outside of Ansible?
- Understand variable precedence
- Reuse, compose
  - Hide complexity in nested templates, macros, plugins
  - Minimize copy/paste and monoliths
- Reshape data where possible
  - Our own facts need a lot of work here!
- Interactively test templates in e.g., ansible-console, ansibug
- New ways to fill in gaps
  - Macros-as-filters/tests
  - Expression/comprehension-esque filters

Questions?

BONUS CONTENT - 2.21 Register Projections

- 2.21 includes new goodies in register
    - Register multiple variables
    - Task conditionals can access the results without using register at all
    - No quotes or {{ }} required (ala set_fact templates)
    - Stacking values (access previous values registered by the same task)
    - For loops, access individual and accumulated loop results at the same time
- DEMO