S107 - Provisioning a CICS Region using Python and Ansible

Drew Hughes
Software Developer – CICS Modernization
Andrew.Hughes1@ibm.com

Disclaimer

Agenda

- What makes CICS configuration hard
- A step wise example of simplifying configuration
- Where Ansible fits into all this
- A more simplified approach?

Challenges facing the next generation of CICS system programmers

- Takes a long time for early tenure system programmers to become comfortable with the platform

- Not something most universities teach

ispf

unix

zos

ebcdic

tso

rexx

jcl

cics

vsam

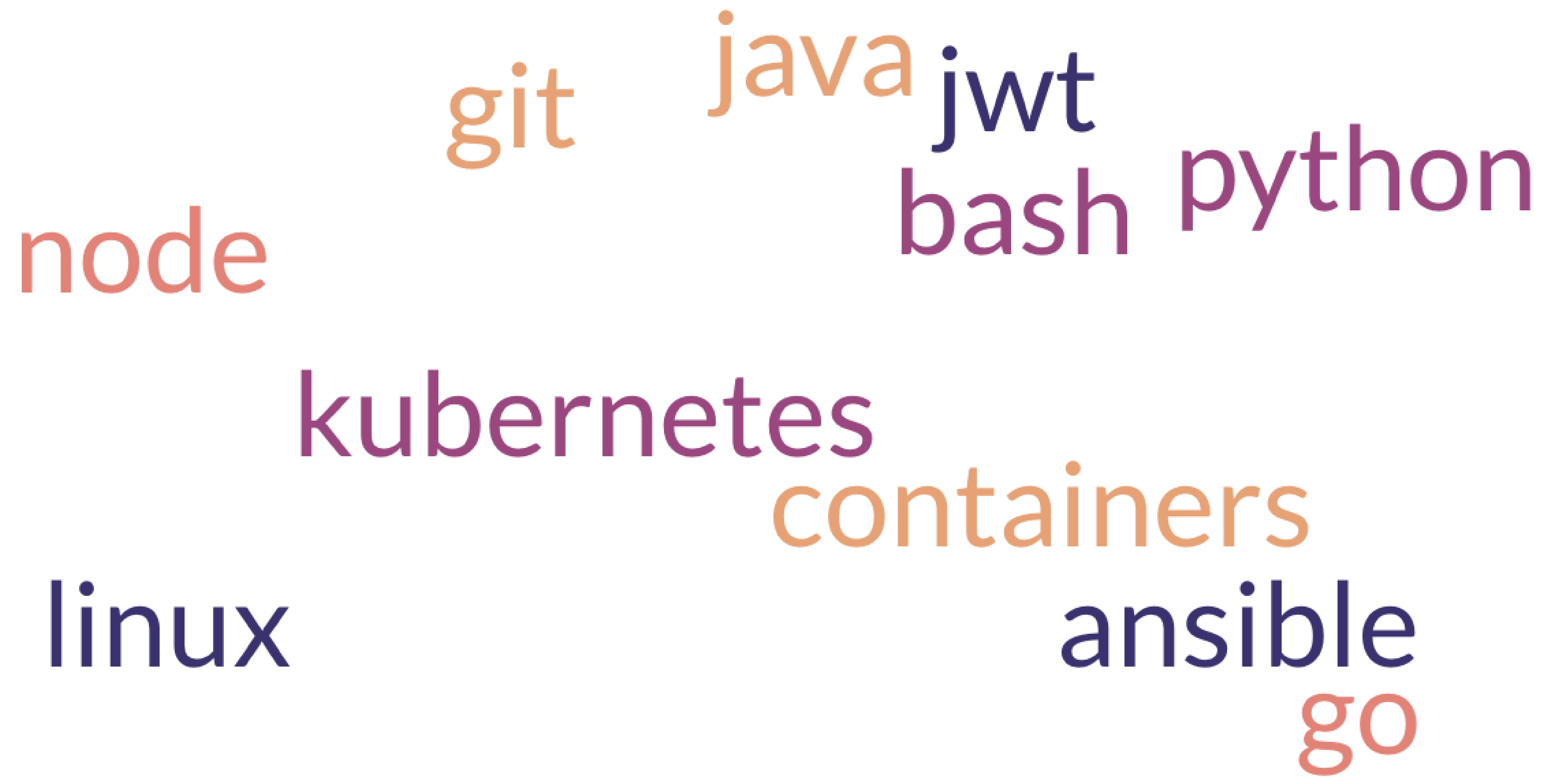
racf

On a more positive note...

- Increasing portfolio of technologies relevant to the platform which the next generation of system programmers will have experienced

- How can we modernize CICS configuration, to make it more familiar to the next gen of system programmers?

- And more powerful, and easier to work with, too!



Why configuration as code?

Move the source of truth from in-situ files and data sets to a system that can reproduce that config on-demand. Ideally you store that in SCM (e.g. git)

Benefits include:

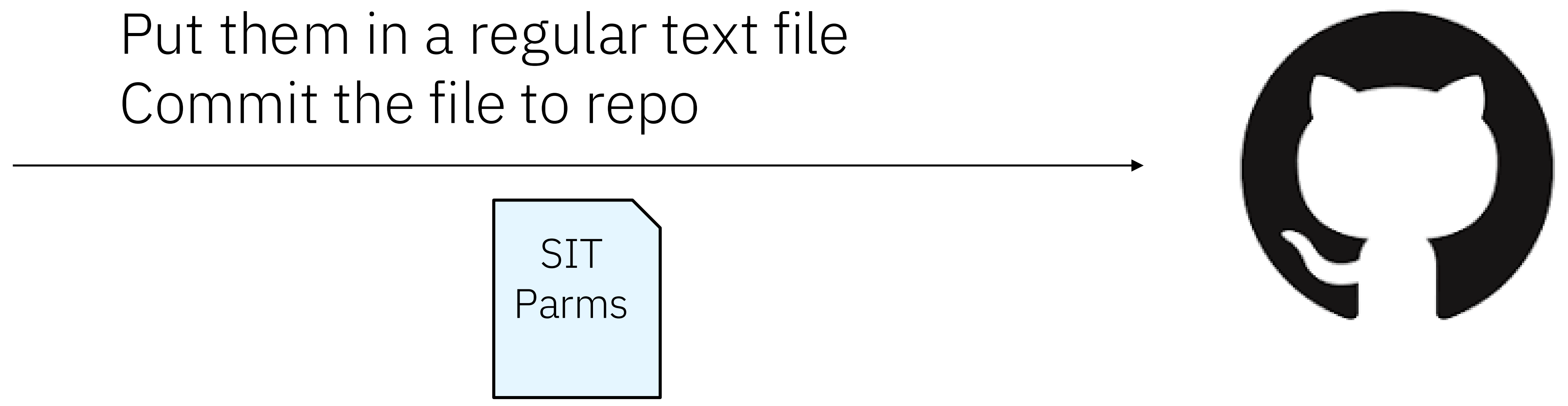
• Audibility	Every change (e.g. JCL, CSD resources, etc) is associate with a reviewed SCM commit
• Consistency	Environments built from the same SCM commit are de-facto consistent
• Reduced risk	All deployments follow the same automated process, no manual intervention
• Faster deployment	Any changes can be deployed in the same way, with an SCM commit
• Back-out	Undo the last commit

A Configuration-as-Code z/OS example

Manual

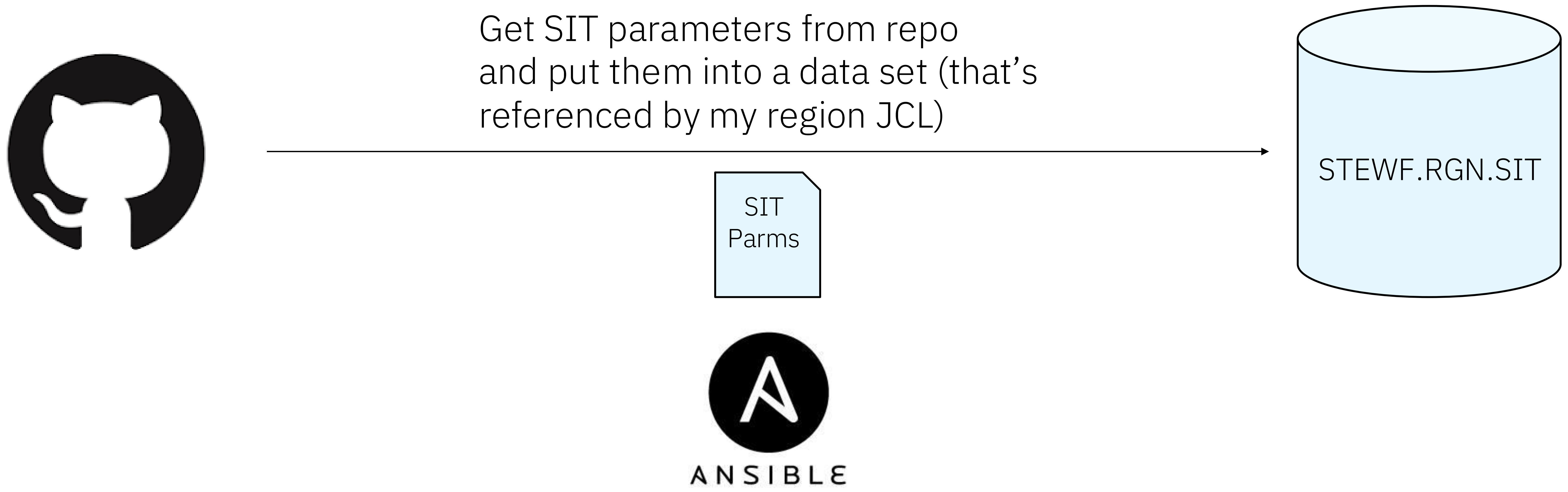
```
APPLID=IYK2ZOE1
SIT=6$
START=INITIAL
CICSSVC=217
GRPLIST=(DFHLIST,A,B)
GMTEXT="Stew's CICS region"
SRBSVC=218
USSHOME=/cics/cics740
SYSIDNT=ZOE1
SEC=YES
```

Take JCL SYSIN SIT parameter overrides
Put them in a regular text file
Commit the file to repo



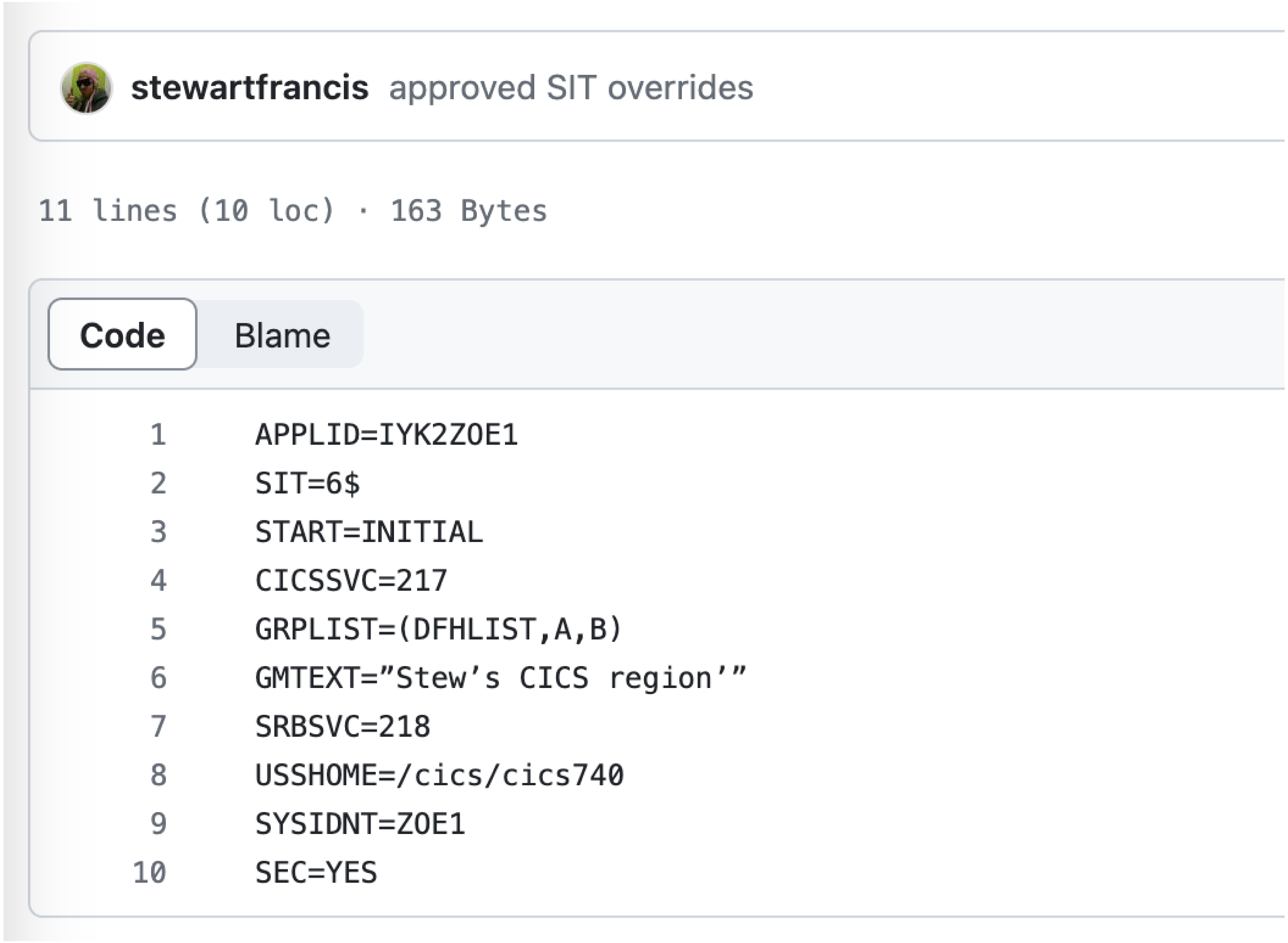
Automated

Get SIT parameters from repo
and put them into a data set (that's
referenced by my region JCL)



What’s the new process for changing my SIT parameters?

- Moved the canonical version of my SIT overrides from z/OS to GitHub
- If anyone wants to change those SIT parameters they can make a **pull request**



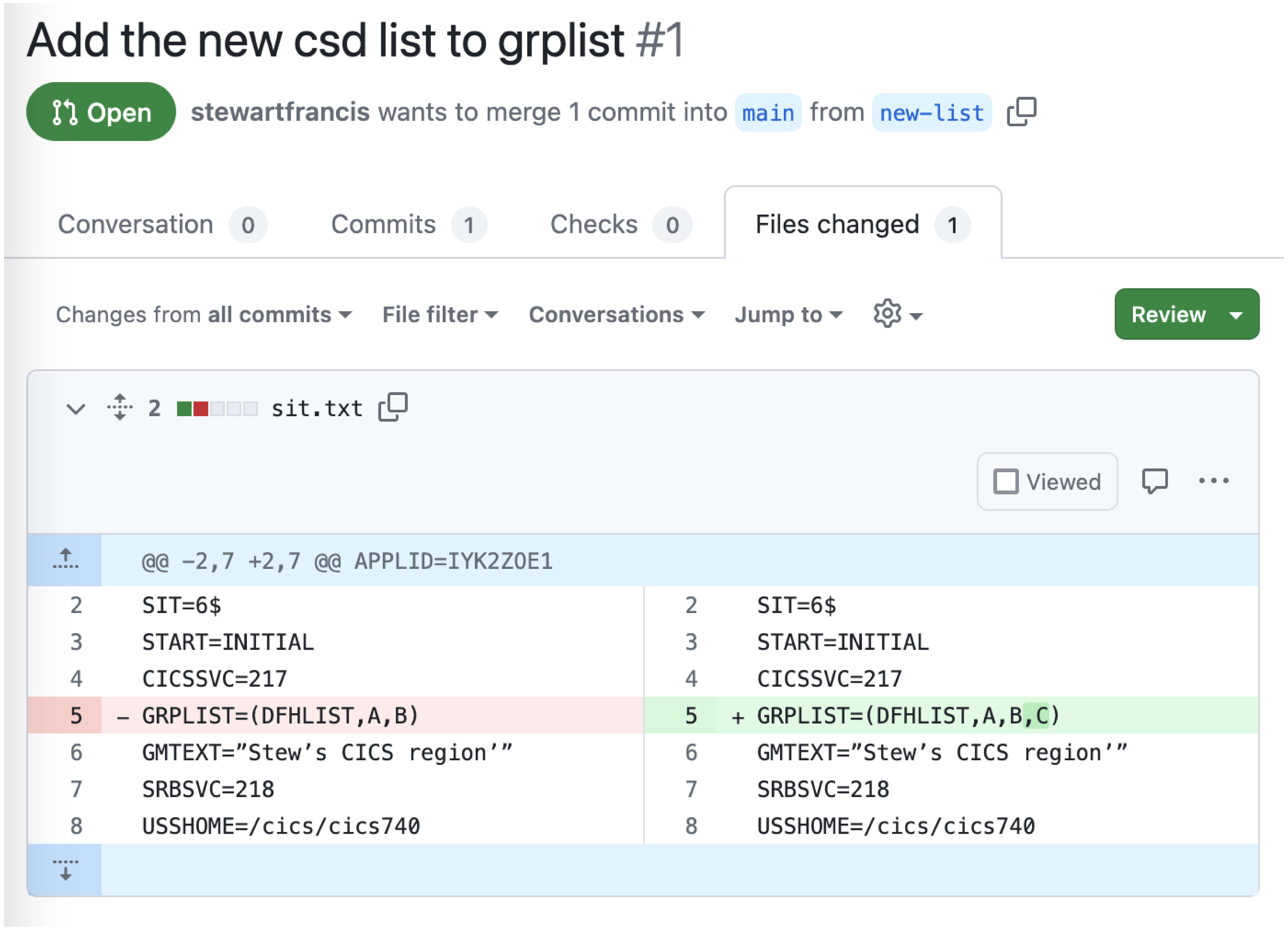
stewartfrancis approved SIT overrides

11 lines (10 loc) · 163 Bytes

```
APPLID=IYK2ZOE1
SIT=6$
START=INITIAL
CICSSVC=217
GRPLIST=(DFHLIST,A,B)
GMTEXT=”Stew’s CICS region’”
SRBSVC=218
USSHOME=/cics/cics740
SYSIDNT=ZOE1
SEC=YES
```

What’s the new process for changing my SIT parameters?

- Changes can be proposed



Add the new csd list to grplist #1

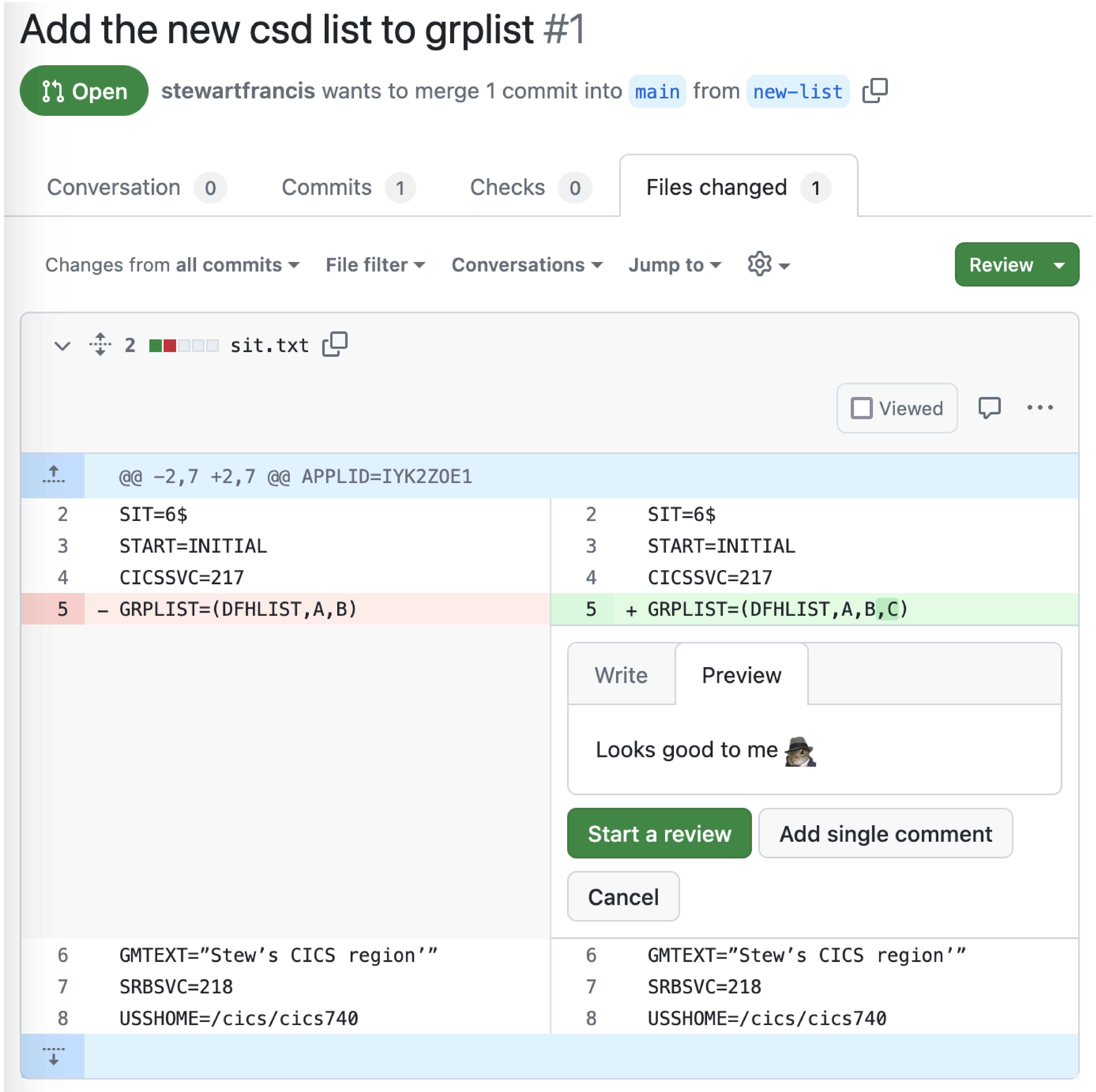
Open stewartfrancis wants to merge 1 commit into main from new-list

2 sit.txt

```
@@ -2,7 +2,7 @@ APPLID=IYK2ZOE1
2   SIT=6$                        2   SIT=6$
3   START=INITIAL                 3   START=INITIAL
4   CICSSVC=217                   4   CICSSVC=217
5 - GRPLIST=(DFHLIST,A,B)         5 + GRPLIST=(DFHLIST,A,B,C)
6   GMTEXT=”Stew’s CICS region’”  6   GMTEXT=”Stew’s CICS region’”
7   SRBSVC=218                    7   SRBSVC=218
8   USSHOME=/cics/cics740         8   USSHOME=/cics/cics740
```

What's the new process for changing my SIT parameters?

- Approved…



Add the new csd list to grplist #1

Open stewartfrancis wants to merge 1 commit into main from new-list

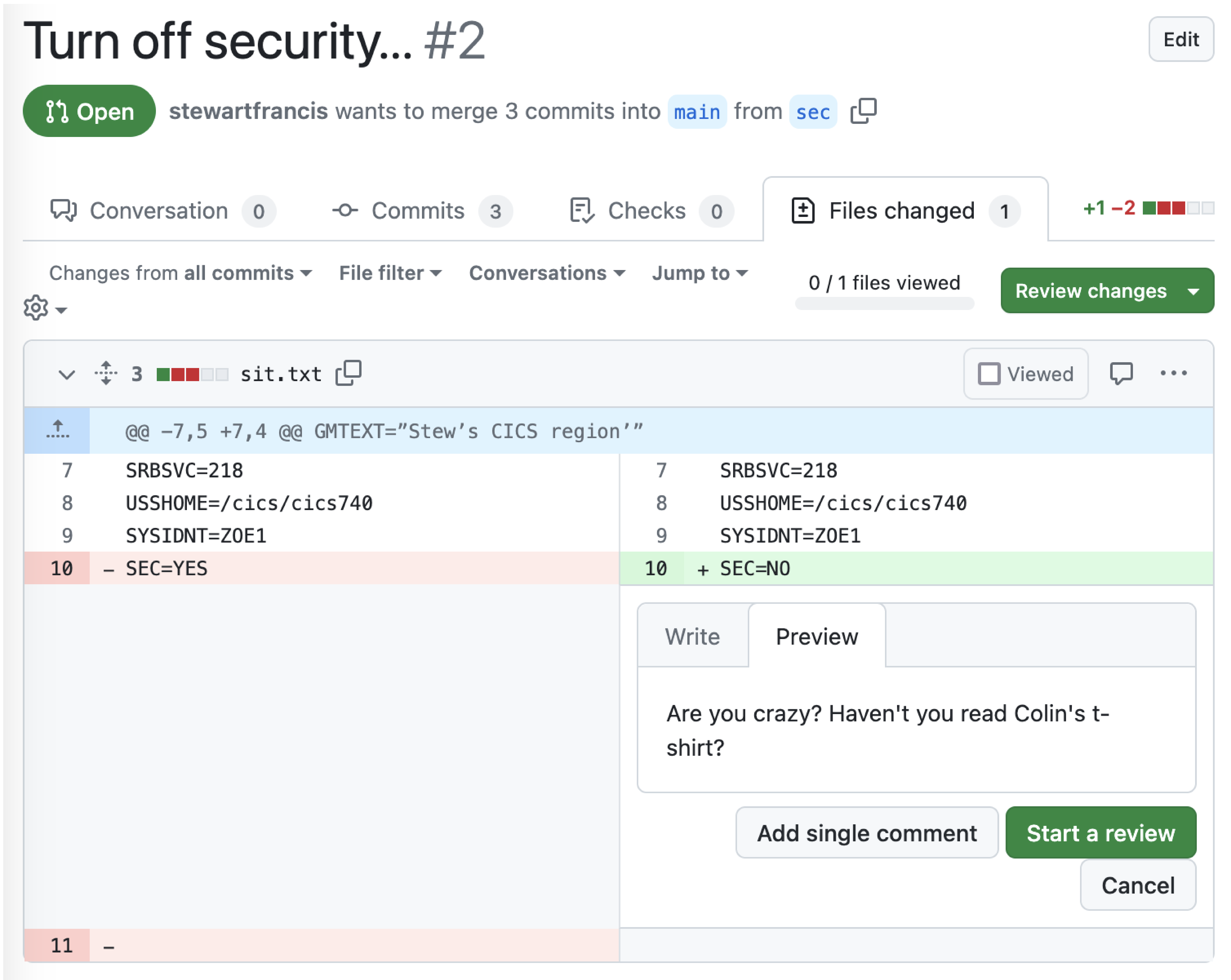
2 sit.txt

```
@@ -2,7 +2,7 @@ APPLID=IYK2ZOE1
 SIT=6$
 START=INITIAL
 CICSSVC=217
-GRPLIST=(DFHLIST,A,B)
+GRPLIST=(DFHLIST,A,B,C)
 GMTEXT="Stew's CICS region'"
 SRBSVC=218
 USSHOME=/cics/cics740
```

Looks good to me

What's the new process for changing my SIT parameters?

- Or rejected…



Turn off security… #2

stewartfrancis wants to merge 3 commits into `main` from `sec`

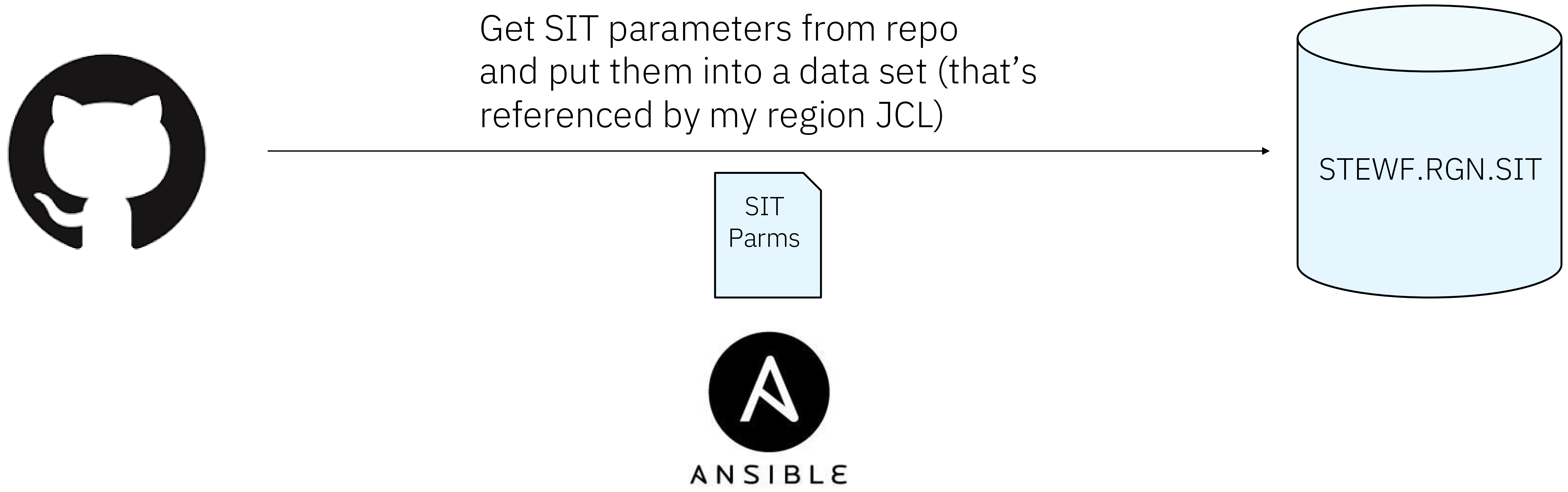
3 sit.txt

```
@@ -7,5 +7,4 @@ GMTEXT="Stew's CICS region'"
7     SRBSVC=218                 7     SRBSVC=218
8     USSHOME=/cics/cics740      8     USSHOME=/cics/cics740
9     SYSIDNT=ZOE1               9     SYSIDNT=ZOE1
10  - SEC=YES                    10  + SEC=NO
11  -
```

Are you crazy? Haven't you read Colin's t-shirt?

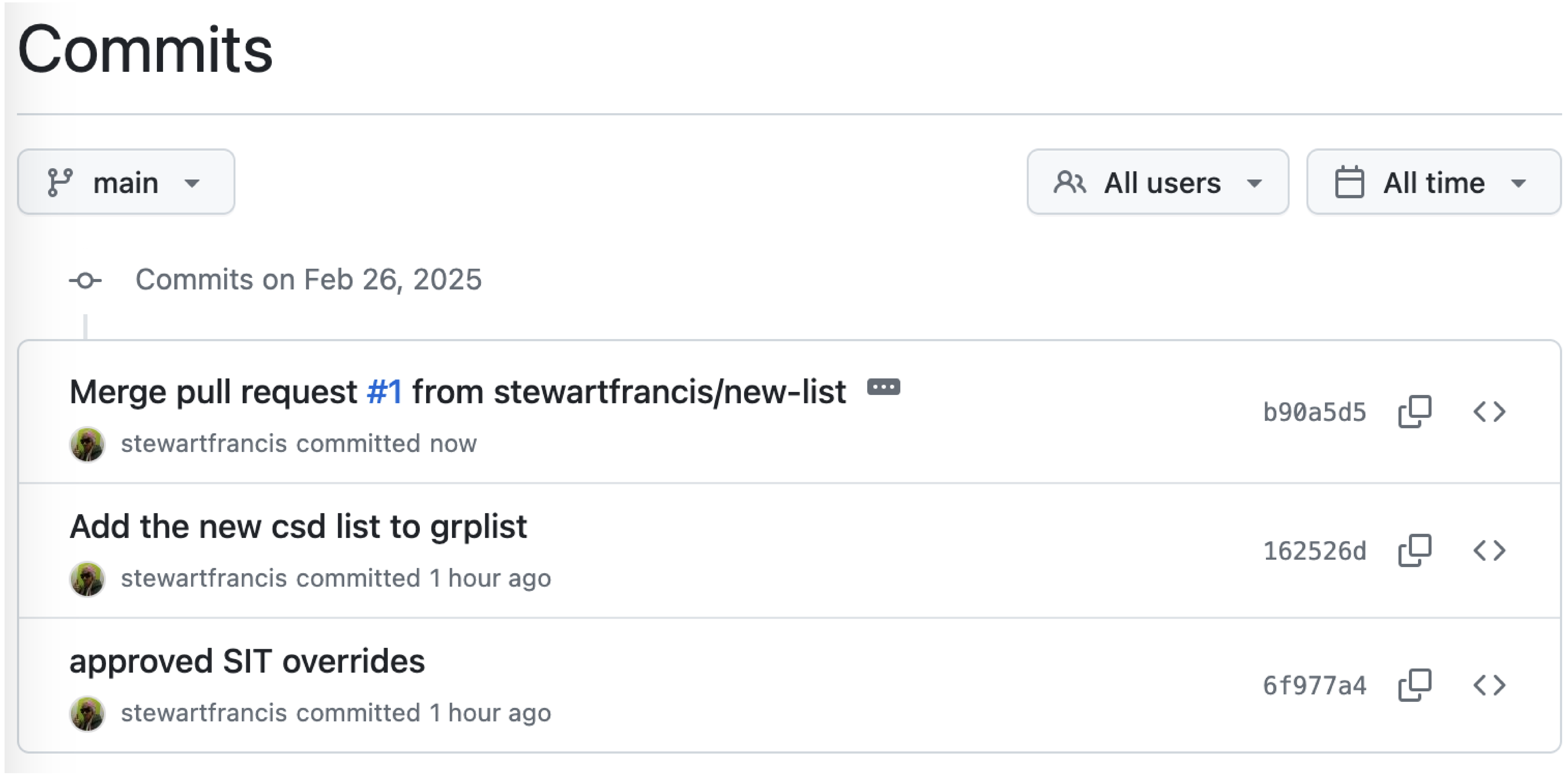
What's the new process for changing my SIT parameters?

- All before the changes are applied to z/OS **at all**
- You can even run automated checks, tests, and scans against the settings
- For instance, ensuring that the settings meet your regulatory requirements
- Once the changes get approved and merged, you can trigger your automated process to make the changes to your system


Get SIT parameters from repo
and put them into a data set (that's referenced by my region JCL)
SIT Parms
ANSIBLE
STEWF.RGN.SIT

What's the new process for changing my SIT parameters?

- This leaves a great paper trail, which is useful for audit purposes
- And you can even revert to an old version of the configuration (and run the automated process to deploy the changes)
- This is a great way of managing risk, and is also a really useful productivity tool – no more change paralysis



Commits on Feb 26, 2025

Merge pull request #1 from stewartfrancis/new-list
stewartfrancis committed now
b90a5d5

Add the new csd list to grplist
stewartfrancis committed 1 hour ago
162526d

approved SIT overrides
stewartfrancis committed 1 hour ago
6f977a4

So what's the problem? Can't we just do this? For all the CICS configuration?

- Clients should go and do this!
  - But it only gets us so far…

- It helps mitigate some of the complexity issues of working with CICS configuration
  - But it doesn't directly address them…

- What can we do to make CICS fundamentally easier to configure and manage?

What makes CICS configuration complex?

- CICS configuration is **diverse**
- Over the years, CICS has adopted a wide array of different technologies, which each come with their own configuration semantics
- Load modules, CICS Web Services, Java applications, Liberty configuration CSD resource definitions, CICS bundles and their varied resources, CPSM configuration, JVM profiles, resource definition overrides, pipeline config files, SIT parameters, SIT overrides, Node.js, etc
- Each type of configuration has its own semantics, and may require special treatment

No in-editor suggestions

No syntax highlighting

```
APPLID=IYK2ZOE1
SIT=6$
START=INITIAL
CICSSVC=217
GRPLIST=(DFHLIST,A,B)
GMTEXT="Drew's CICS region"
SRBSVC=218
USSHOME=/cics/cics740
SYSIDNT=ZOE1
SEC=YES
```

How does line wrapping work?

No validation

No integrated doc

What about all the other configuration file formats?

What makes CICS configuration complex?

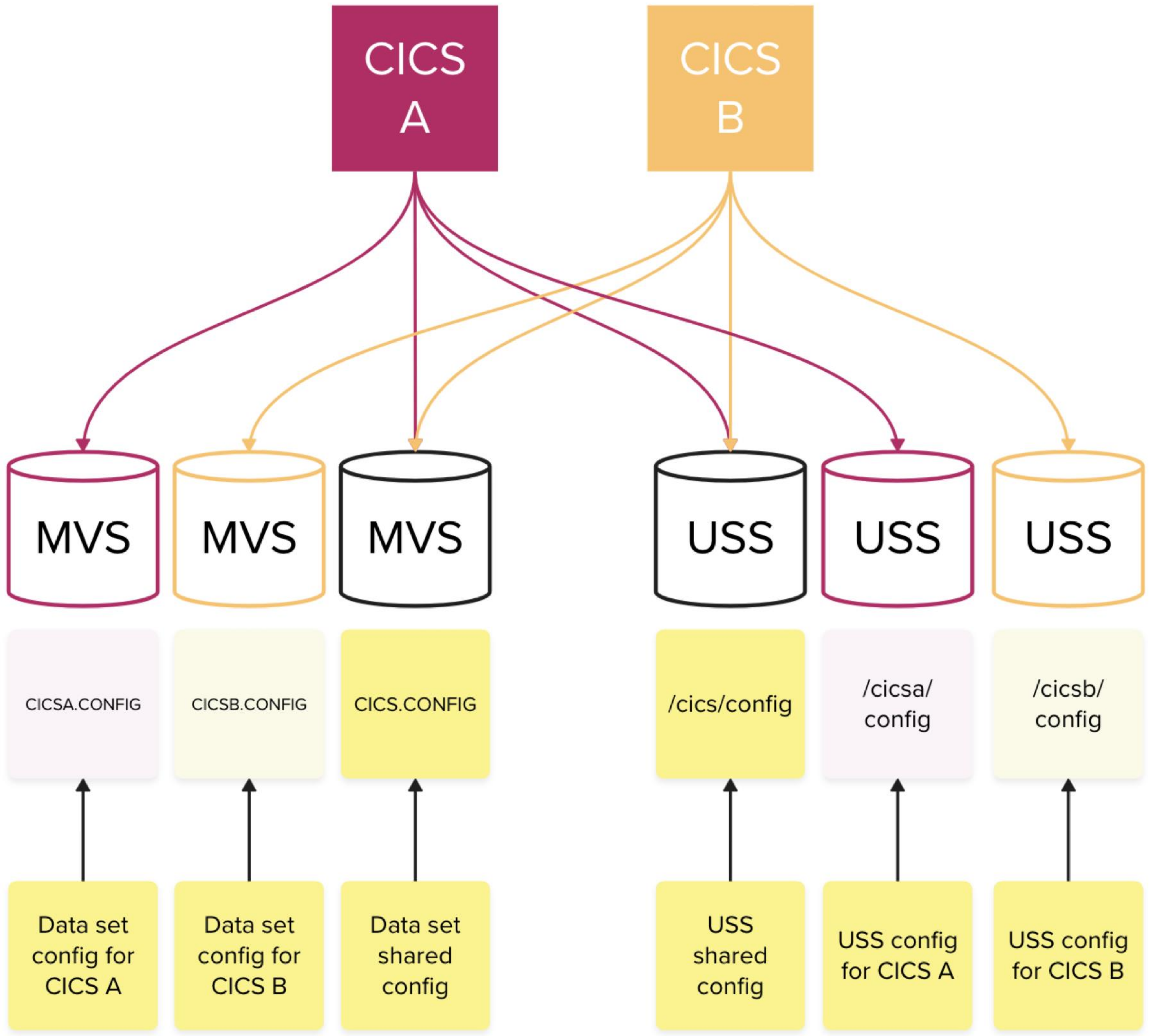
- CICS configuration is **shared**

- When you configure CICS, you're not typically configuring an individual system

- Ordinarily, some configuration is shared between CICS regions
  - SIT parameters
  - Load libraries
  - CSD resources

- Whilst other configuration needs to remain distinct
  - Ports
  - Applids
  - CSD resources
  - Load libraries
  - Server.xml
  - SIT parameters
  - Etc

- This mechanism is **bespoke** for each client. There's not a common configuration management system

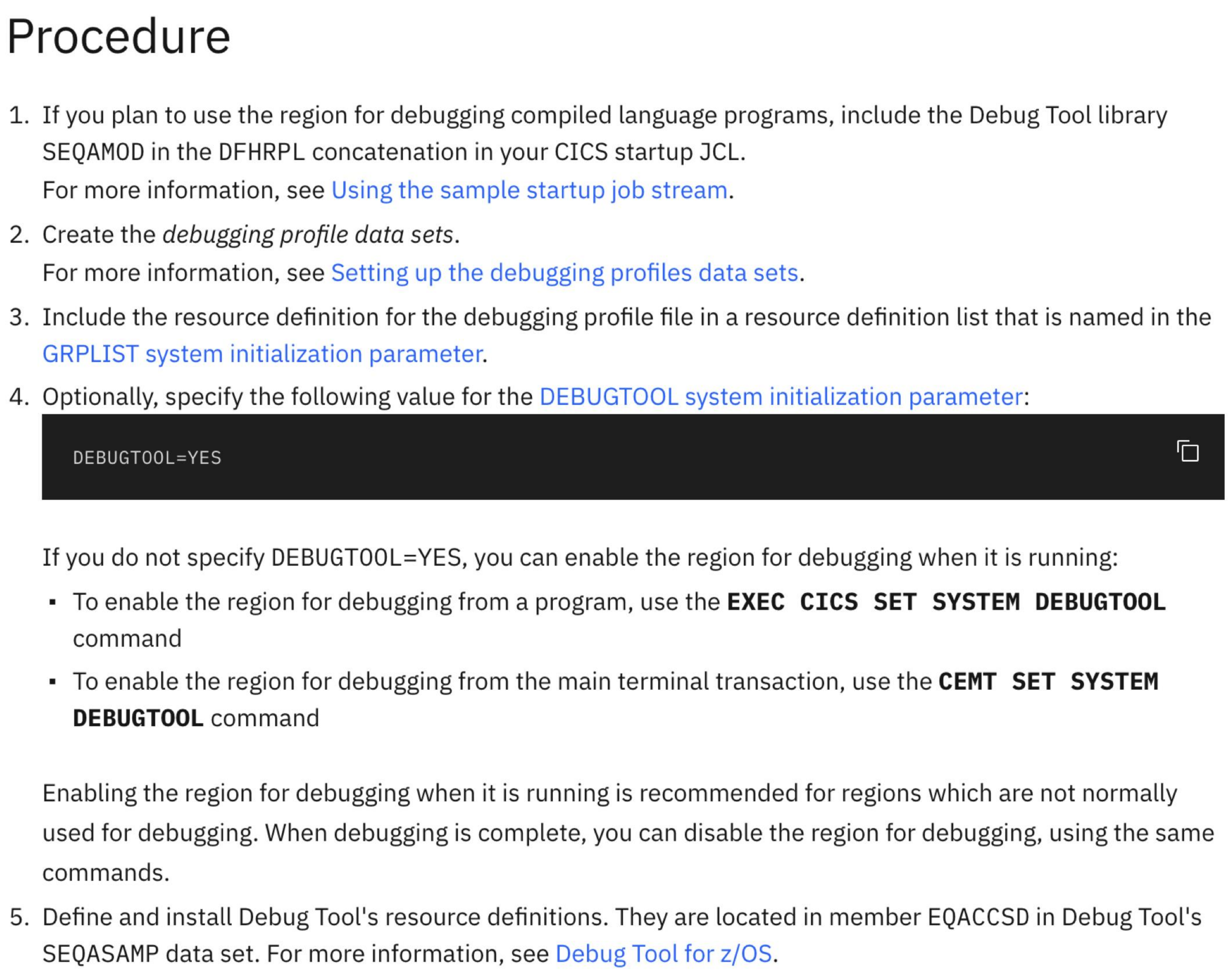


What makes CICS configuration complex?

There's **little off-the-shelf configuration**

- Where is IBM Debug installed?
- Where do you want to create your debugging profiles data sets?
- Where is the JCL we should add the DDs to?
- Where is your CSD?
- Which CSD Lists should we add the definitions to?
- Do you want to enable debugging on start-up?

- This is a simple example!



Procedure

1. If you plan to use the region for debugging compiled language programs, include the Debug Tool library SEQAMOD in the DFHRPL concatenation in your CICS startup JCL.
   For more information, see Using the sample startup job stream.
2. Create the *debugging profile data sets*.
   For more information, see Setting up the debugging profiles data sets.
3. Include the resource definition for the debugging profile file in a resource definition list that is named in the GRPLIST system initialization parameter.
4. Optionally, specify the following value for the DEBUGTOOL system initialization parameter:

   ```
   DEBUGTOOL=YES
   ```

   If you do not specify DEBUGTOOL=YES, you can enable the region for debugging when it is running:
   - To enable the region for debugging from a program, use the **EXEC CICS SET SYSTEM DEBUGTOOL** command
   - To enable the region for debugging from the main terminal transaction, use the **CEMT SET SYSTEM DEBUGTOOL** command

   Enabling the region for debugging when it is running is recommended for regions which are not normally used for debugging. When debugging is complete, you can disable the region for debugging, using the same commands.
5. Define and install Debug Tool's resource definitions. They are located in member EQACCSD in Debug Tool's SEQASAMP data set. For more information, see Debug Tool for z/OS.

What can we do about it?

$$\text{“}\underbrace{\text{Simplify the configuration of CICS}}_{1}\ \underbrace{\text{using a declarative configuration language}}_{2}\text{”}$$

What are our simplification goals?

- A unified configuration system, encompassing all CICS configuration
- Suitable for storing in a source code management system
- With a great editing experience
- Providing the opportunity to deliver high-level configuration options

Why is “declarative” configuration important?

Procedural information describes “how” to do something
- turn-by-turn directions
- Cooking recipes

Declarative information describes a state
- A desired destination and arrival time
- A menu

Declarative configuration…
- ... is easier to reason about
- ... more easily auditable
- ... offers the opportunity for off-the-shelf automation
- ... can be repeatedly applied to the same system safely

CICS Transaction Server open beta Program

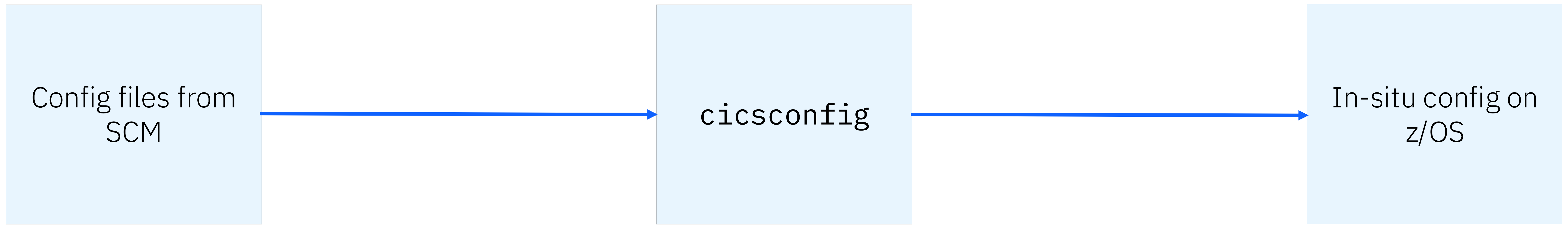
CICS TS open beta

Sample tool to configure CICS Regions version 0.3.0

If running in an air-gapped environment (with no access to the internet), please use the zip file that includes all the dependencies. Otherwise you can use the wheel on its own. For more information, look at the README.pdf

File Description	**File Name (Click on to Download)**	**File Size**
Sample Tool	cicsconfig-0.3.0.zip	2.1 MB
Sample Tool wheel only	cicsconfig-0.3.0-py3-none-any.whl	62 KB
CICS Region Schema	cics_region_0.3.0.schema.json	85 KB
Release Notes	release-notes.pdf	25 KB
Documentation	README.pdf	110 KB
Notices	Notices.txt	20 KB

High-level architecture



YAML-based config language +
Other assets like:
- Resource definition yaml
- Resource definition overrides
- CICS bundles
- JVM profiles
- Liberty server.xml

Python configuration processor tool
- Runs directly on z/OS
- Uses ZOAU APIs
- Translates input configuration into data sets and USS files

- CICS Datasets: CSD, GCD, LCD, Aux trace, Dump, Intra, LRQ, Temp Storage
- Region JCL
- JVM profile directory
- CICS bundles
- etc

Demos Part 1

A generic platform for z/OS configuration management

The framework we are building is not CICS-specific

– A configuration system, that makes it easy to translate YAML-based configuration into data sets, unix files, etc

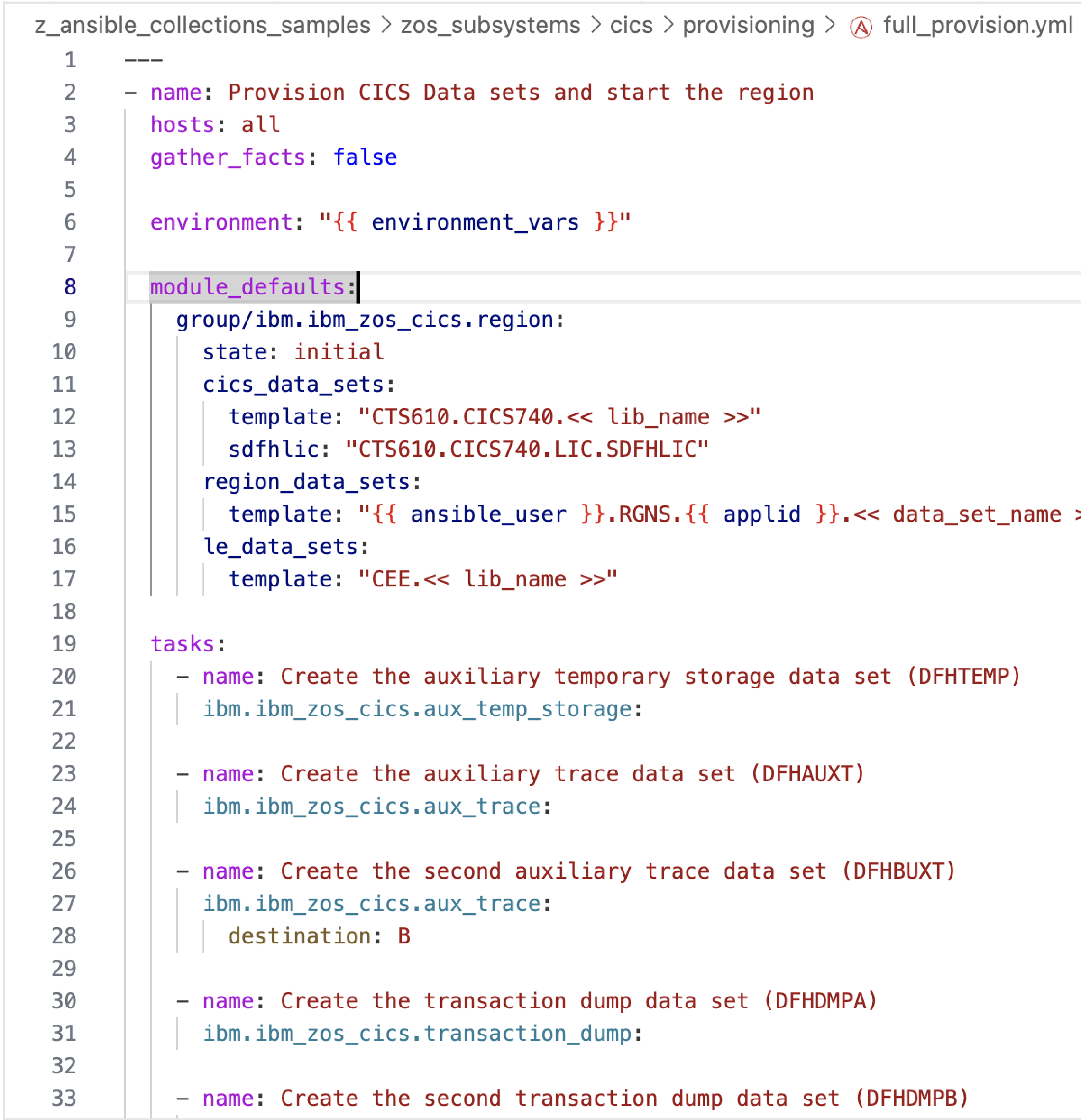
The CICS configuration language is an **implementation** on top of that generic foundation

(Desired) Features

- Built-in validation against a schema
- Editor support
- Declarative state management
- Variables
- Compose configuration from multiple sources
- Extensibility via configuration modules
- Library of utility functions for common configuration management activities

Ansible is a declarative configuration-as-code system. Is that the answer?

- It's part of the answer…
- Each Ansible task is capable of behaving declaratively
  - Many Ansible tasks are declarative (e.g. `yum`)
  - But it depends on the task implementation
- A playbook itself is still procedural
- We don't have the right vocabulary to build a declarative configuration system for CICS inside Ansible

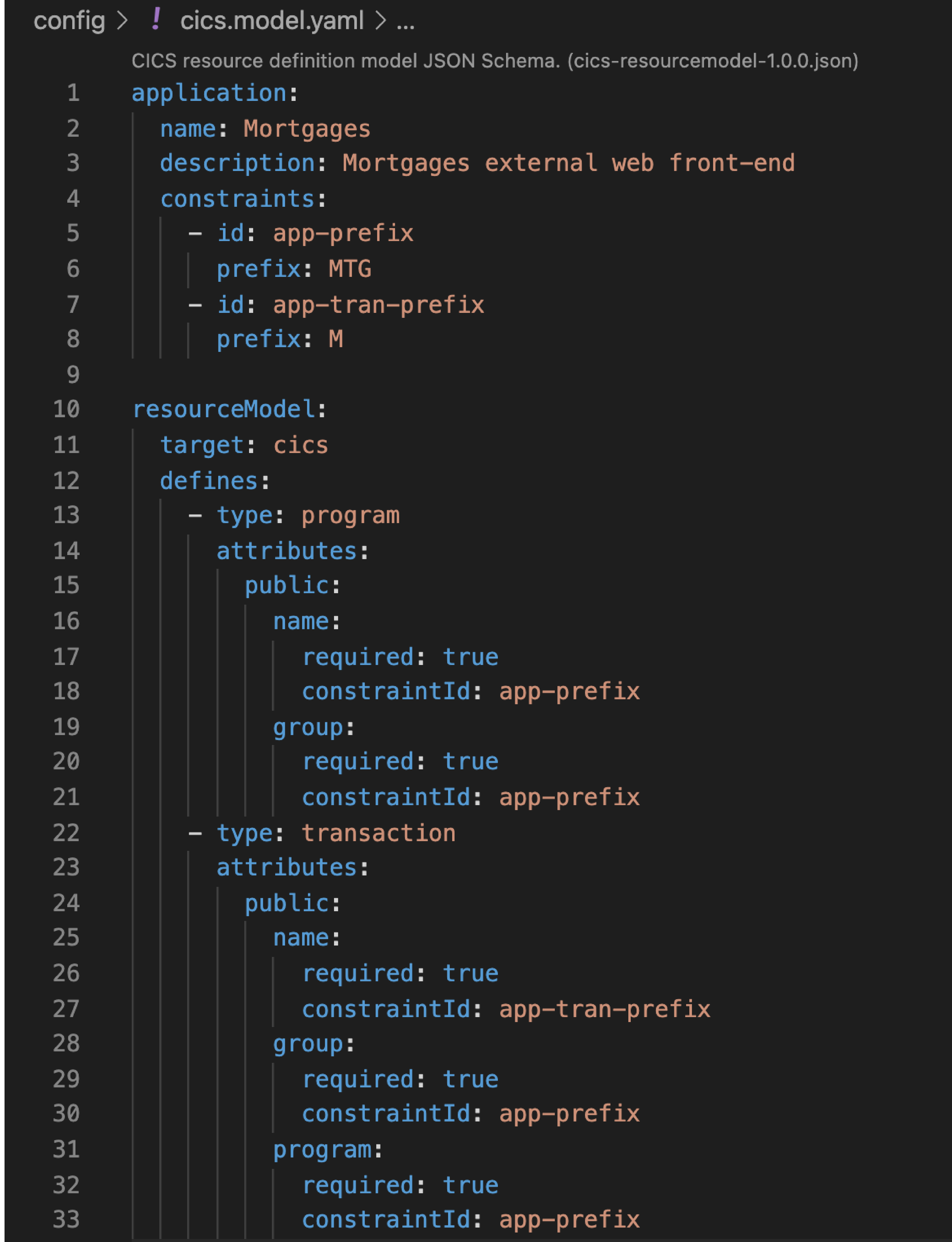

z_ansible_collections_samples > zos_subsystems > cics > provisioning > full_provision.yml

```yaml
---
- name: Provision CICS Data sets and start the region
  hosts: all
  gather_facts: false

  environment: "{{ environment_vars }}"

  module_defaults:
    group/ibm.ibm_zos_cics.region:
      state: initial
      cics_data_sets:
        template: "CTS610.CICS740.<< lib_name >>"
        sdfhlic: "CTS610.CICS740.LIC.SDFHLIC"
      region_data_sets:
        template: "{{ ansible_user }}.RGNS.{{ applid }}.<< data_set_name >
      le_data_sets:
        template: "CEE.<< lib_name >>"

  tasks:
    - name: Create the auxiliary temporary storage data set (DFHTEMP)
      ibm.ibm_zos_cics.aux_temp_storage:

    - name: Create the auxiliary trace data set (DFHAUXT)
      ibm.ibm_zos_cics.aux_trace:

    - name: Create the second auxiliary trace data set (DFHBUXT)
      ibm.ibm_zos_cics.aux_trace:
        destination: B

    - name: Create the transaction dump data set (DFHDMPA)
      ibm.ibm_zos_cics.transaction_dump:

    - name: Create the second transaction dump data set (DFHDMPB)
```

CICS Resource Builder

- CICS resource builder is a configuration-as-code tool from the CICS TS team for managing resource definitions as yaml documents

- System programmers specify a **model** document, which describes which attributes an application developer can control, and establishes constraints on which values they can have


config > ! cics.model.yaml > ...
CICS resource definition model JSON Schema. (cics-resourcemodel-1.0.0.json)

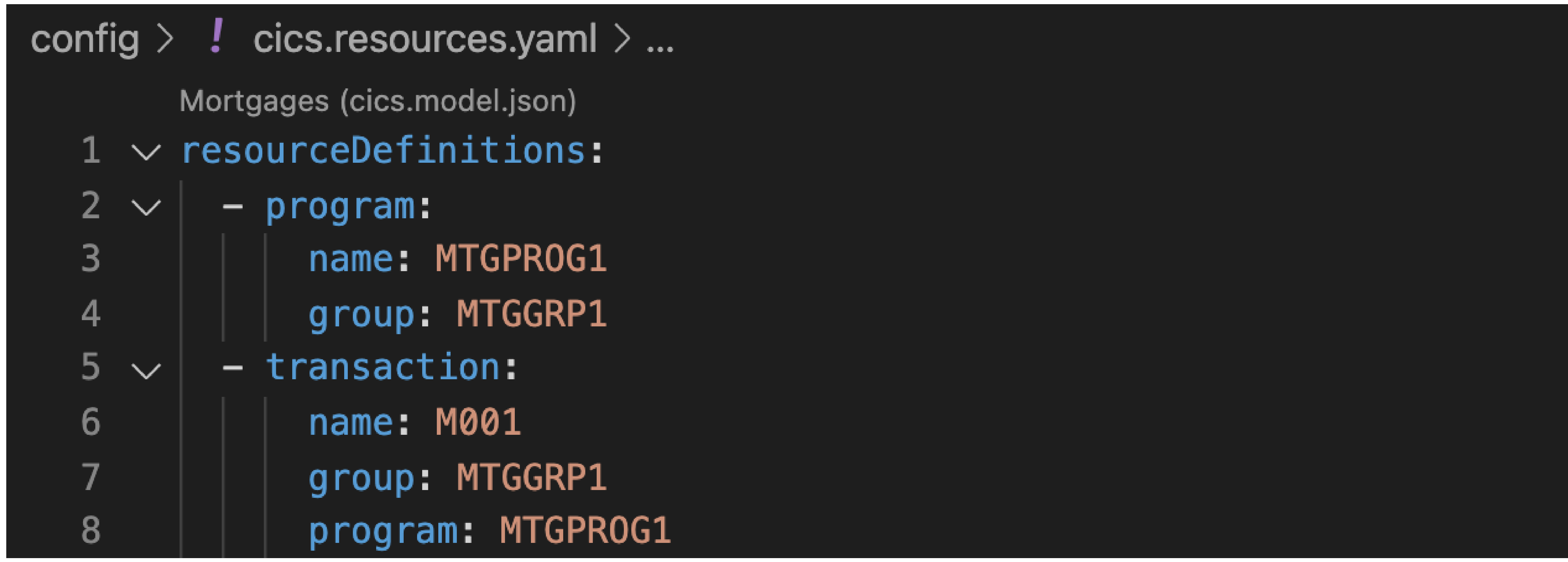
```yaml
application:
  name: Mortgages
  description: Mortgages external web front-end
  constraints:
    - id: app-prefix
      prefix: MTG
    - id: app-tran-prefix
      prefix: M

resourceModel:
  target: cics
  defines:
    - type: program
      attributes:
        public:
          name:
            required: true
            constraintId: app-prefix
          group:
            required: true
            constraintId: app-prefix
    - type: transaction
      attributes:
        public:
          name:
            required: true
            constraintId: app-tran-prefix
          group:
            required: true
            constraintId: app-prefix
          program:
            required: true
            constraintId: app-prefix
```

CICS Resource Builder

- Application developers can then author **definition** files, subject to the constraints provided by the system programmer

- The `zrb` tool can be used to:
  - Generate a schema from the **model** which can be used to validate the **definitions**

  - Use the **model** and the **definitions** to generate input for the DFHCSDUP utility program


config > ! cics.resources.yaml > ...

Mortgages (cics.model.json)

```yaml
resourceDefinitions:
  - program:
      name: MTGPROG1
      group: MTGGRP1
  - transaction:
      name: M001
      group: MTGGRP1
      program: MTGPROG1
```

zrb build –m model.yaml –r defs.yaml –o csdup.txt

Cicsconfig support for Resource Builder

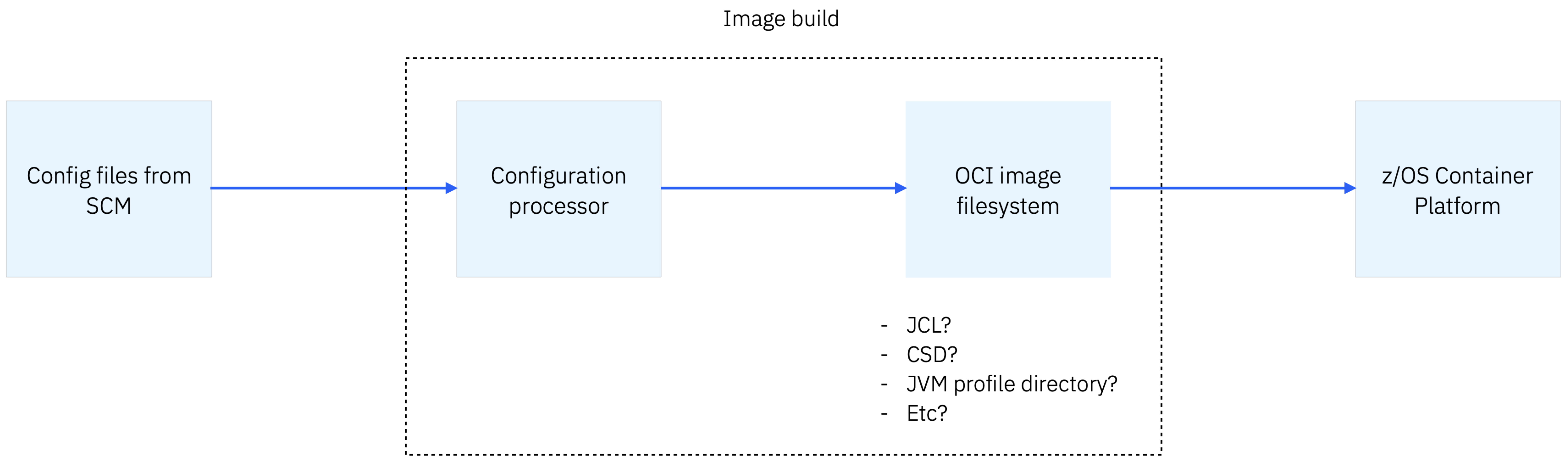
The `cicsconfig` tool supports integrated Resource Builder documents

… and can automatically add their content to the CSD at configuration-time

Let’s look at a few demos of that in action

Demos Part 2

How does this relate to z/OS Containers?



- Cicsconfig tool might make a lot of sense to use as part of your container image build
- Any client investment in config simplification technology will make z/OS container platform easier to adopt in the future

Thoughts for the future…

- Better editor support without having to pass around schemas?
- More artifacts supported in the config files?
- User-extensions to the config language?
- More exhaustive variable support?
- Multiple CICS regions?
- CICSplex?
- Compose configuration from multiple files?
- Onboarding assistant?
- Dry-run?
- Add config to an existing CICS region?
- Incremental update?

Try it out

Give the “sample tool to configure CICS regions”… a try:

https://www.ibm.com/support/pages/ibm-cics-transaction-server-zos-open-beta-program

If you’re interested in providing feedback and hearing about our direction, consider joining the CICS Design Partnership

IBM

GSE GUIDE SHARE EUROPE

Nordic Region

ATTENDEE FEEDBACK IS IMPORTANT!

Please complete a session evaluation for each session you attend

S107

https://gse-nordic.org/session-evaluation