Provisioning a CICS Region using Python and Ansible: Harnessing the power of code

Andrew Hughes
Software Developer, CICS Modernization
andrew.hughes1@ibm.com

IBM

Disclaimer

IBM's statements regarding its plans, directions and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

IBM Confidential. Unless specifically advised otherwise, you should assume that all the information presented in CICS Design Forum sessions and contained in any presentations (whether given in writing or orally) is IBM Confidential and restrict access to this information in accordance with the IBM Feedback Agreement.

Content Authority. The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

Performance. Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Customer Examples. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Availability. References in CICS Design Forum sessions and any presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

Trademarks. IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at http://www.ibm.com/legal/copytrade.shtml

Agenda

– Why is CICS configuration complex?

– What people are doing today

– How could we simplify it?

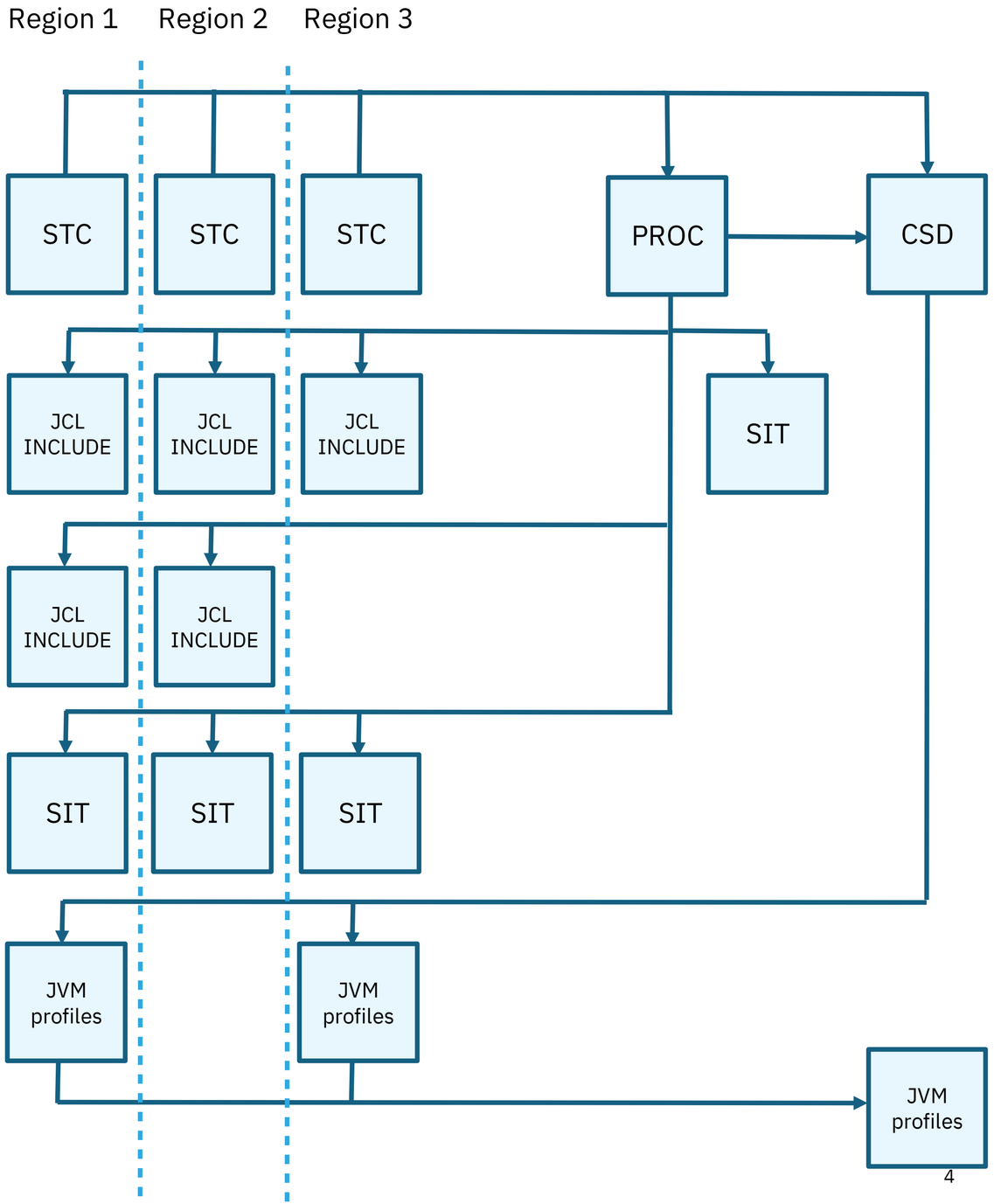
– Why should you care?

– How could I automate it?

What’s wrong with CICS configuration

Very complex and suffers from many of these flaws:

- Lots of files
- Lots of formats
- Data spread in many locations
- Convoluted structure
- Lack of clarity of parameters
- Lack of assistance



How some clients are solving this today

A Configuration-as-Code z/OS example

Manual

```
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
```

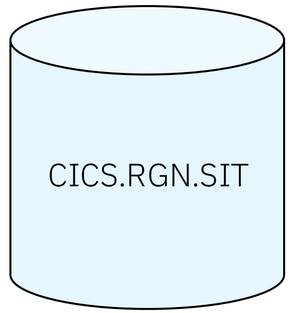
Take JCL SYSIN SIT parameter overrides
Put them in a regular text file
Commit the file to repo

SIT Parms

Automated

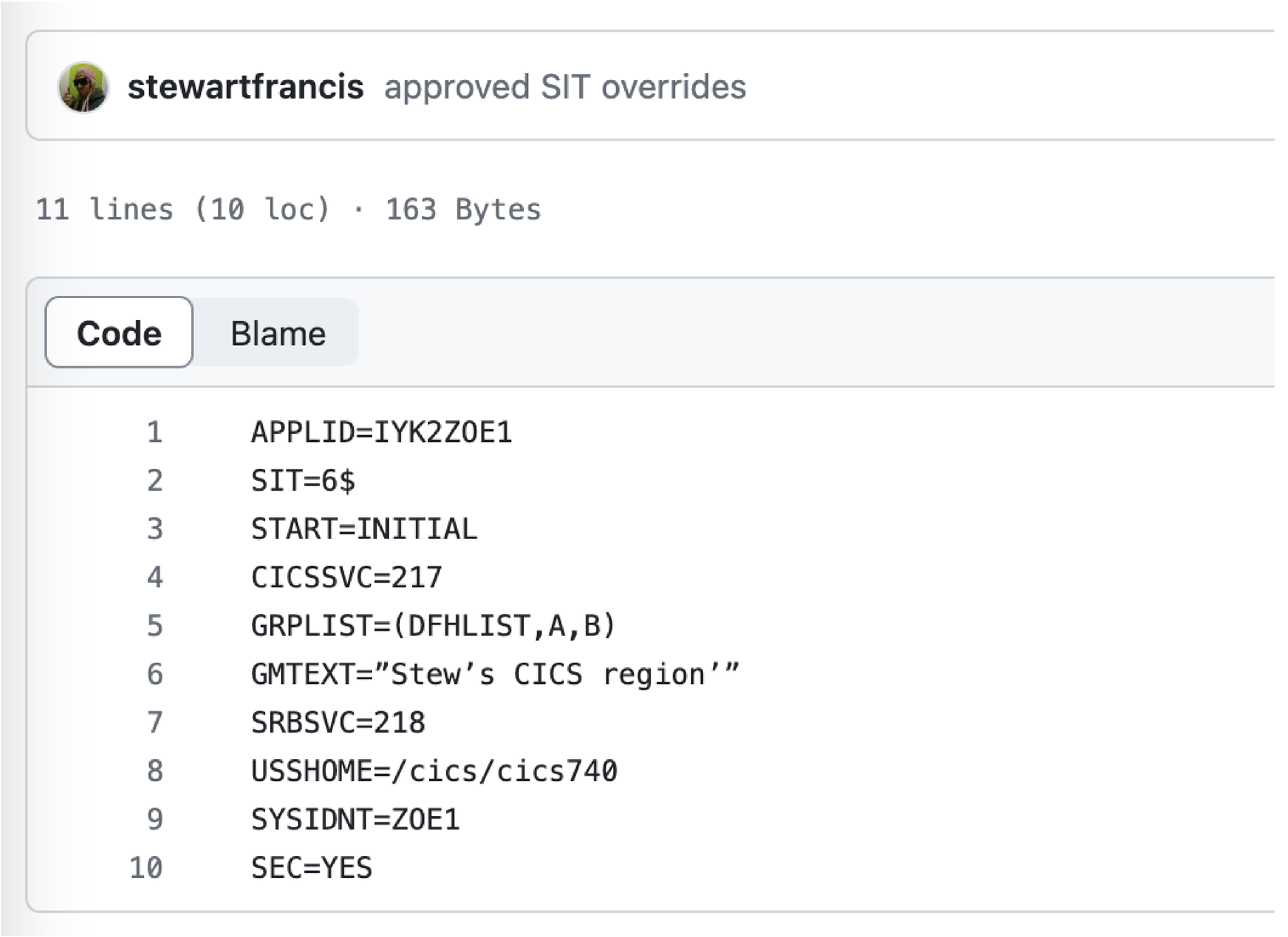
Get SIT parameters from repo
and put them into a data set (that's referenced by my region JCL)

SIT Parms



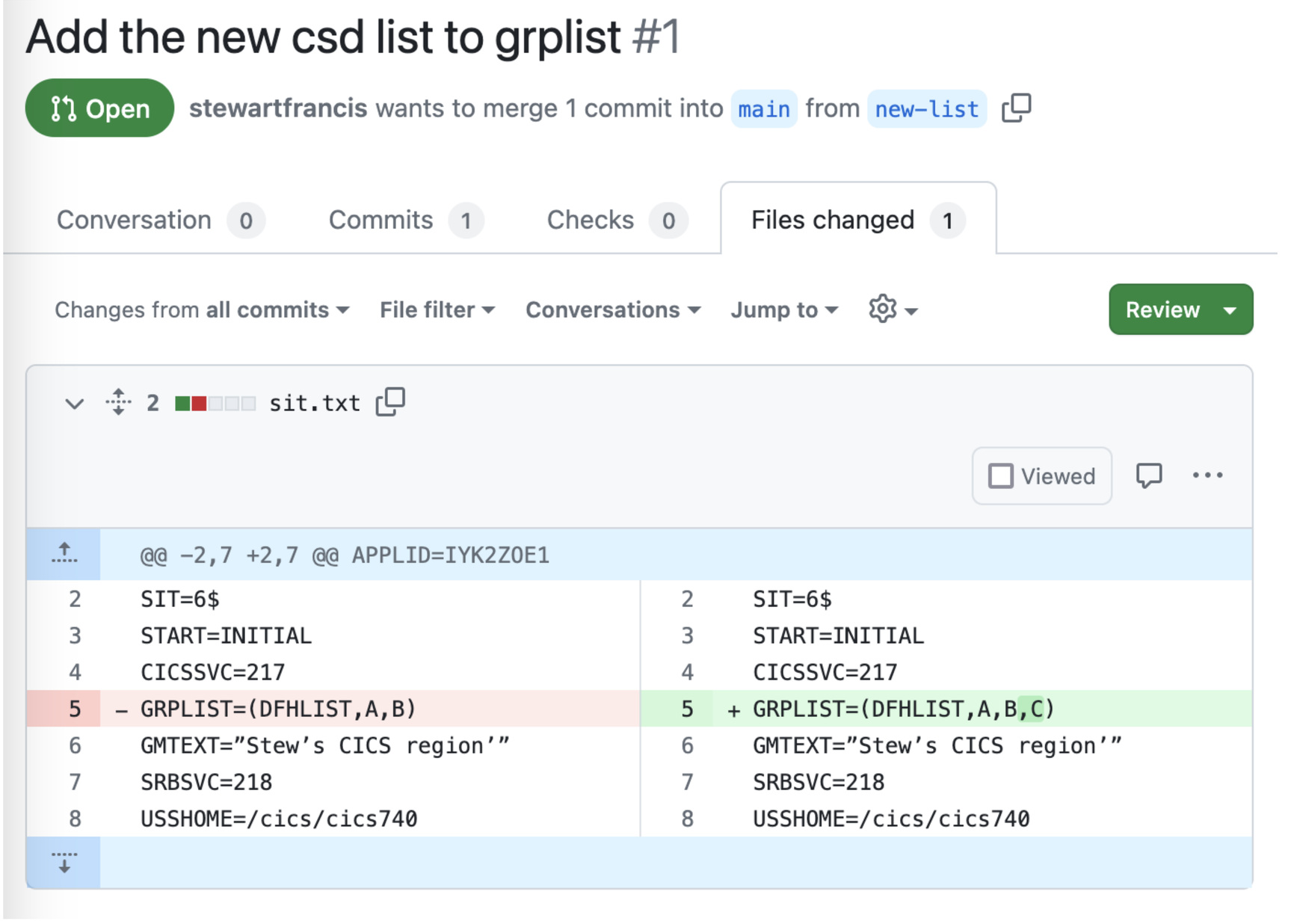
What’s the new process for changing my SIT parameters?

- 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**



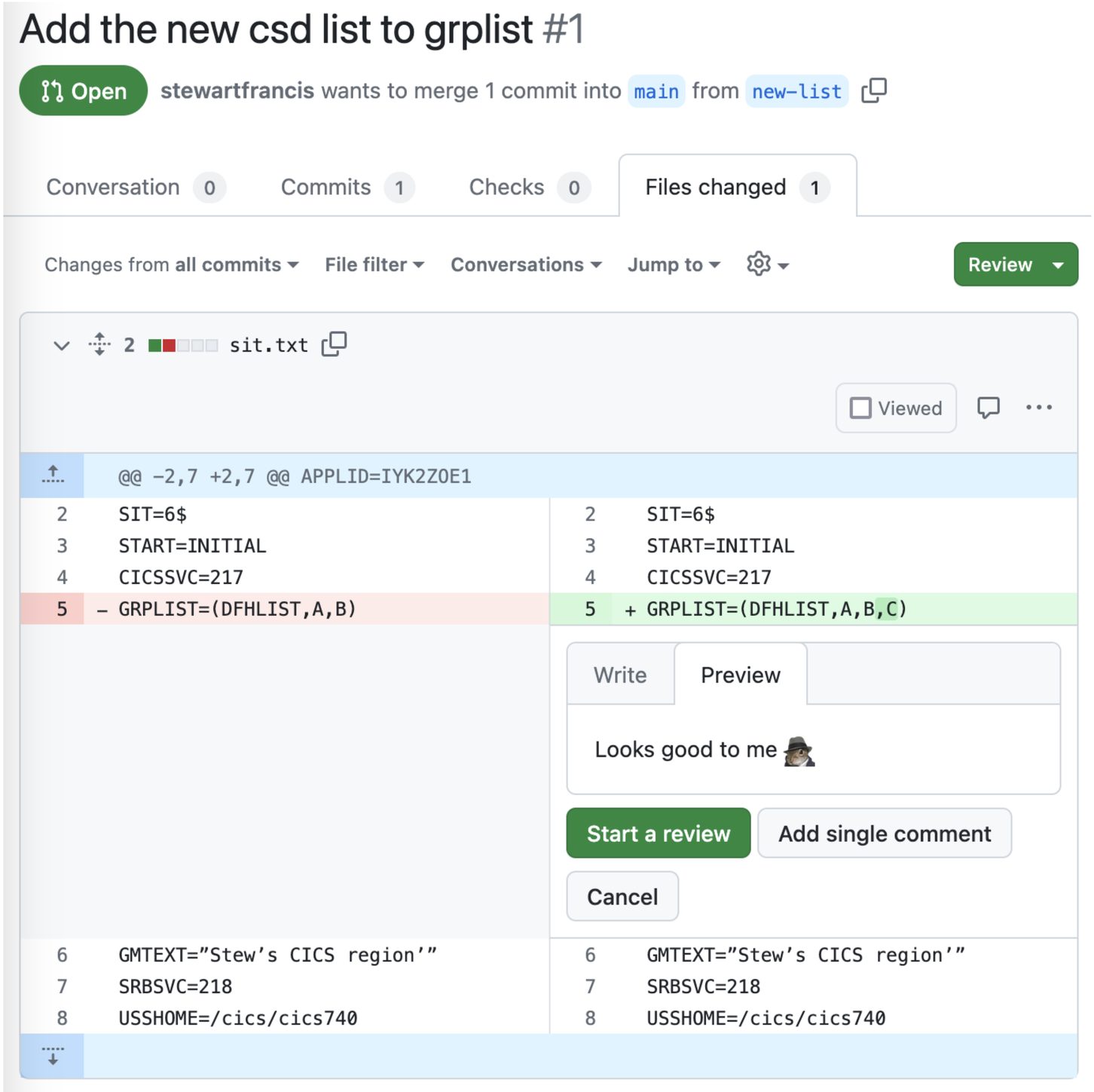
What’s the new process for changing my SIT parameters?

- Changes can be proposed



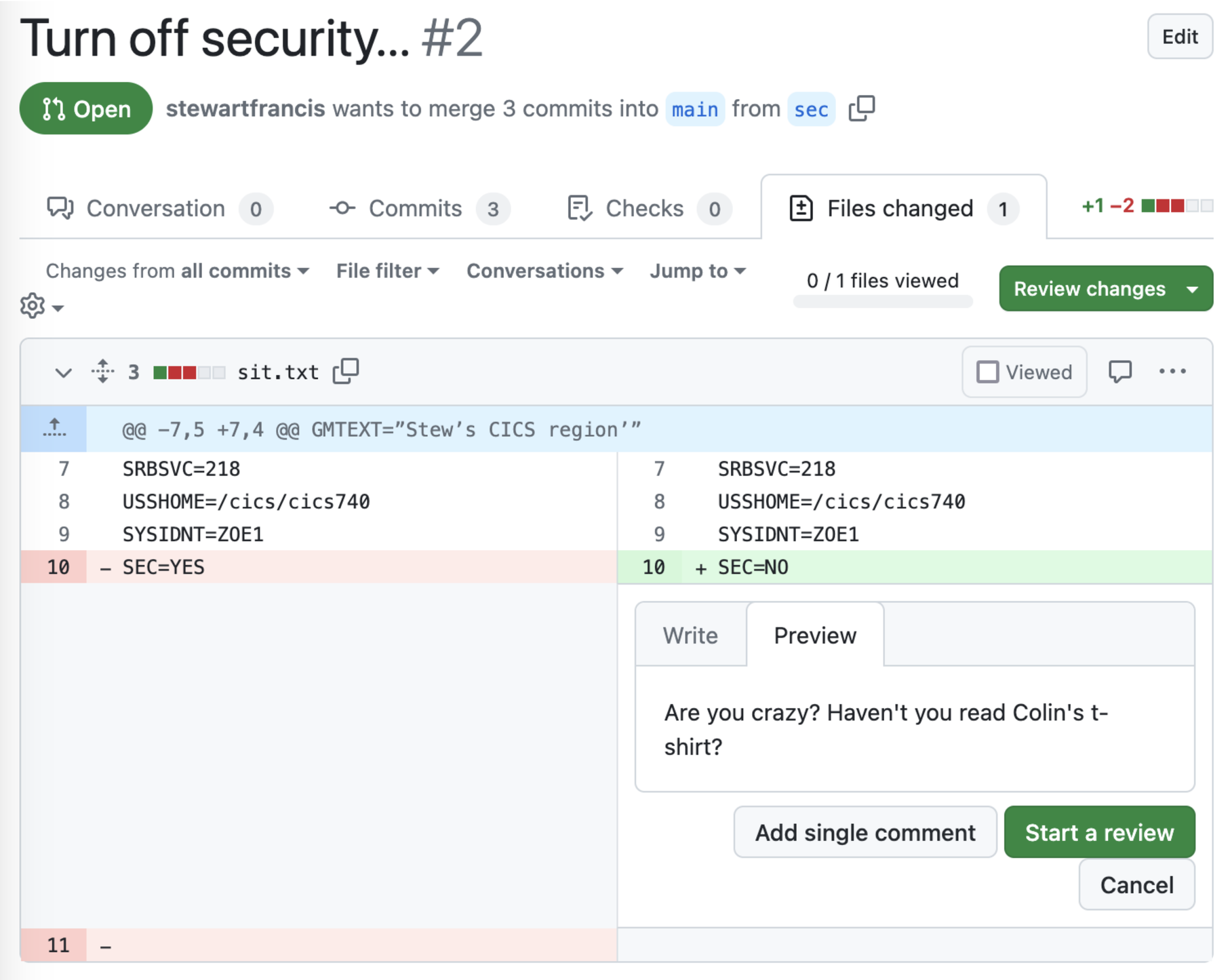
What’s the new process for changing my SIT parameters?

- Approved…



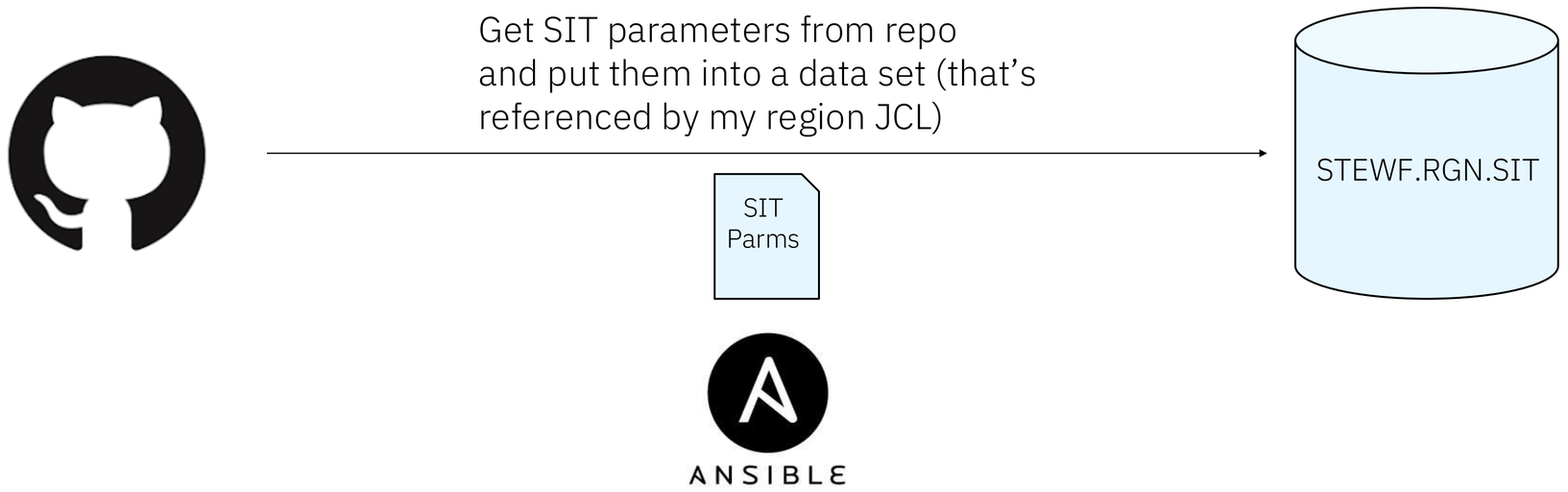
What's the new process for changing my SIT parameters?

- Or rejected...



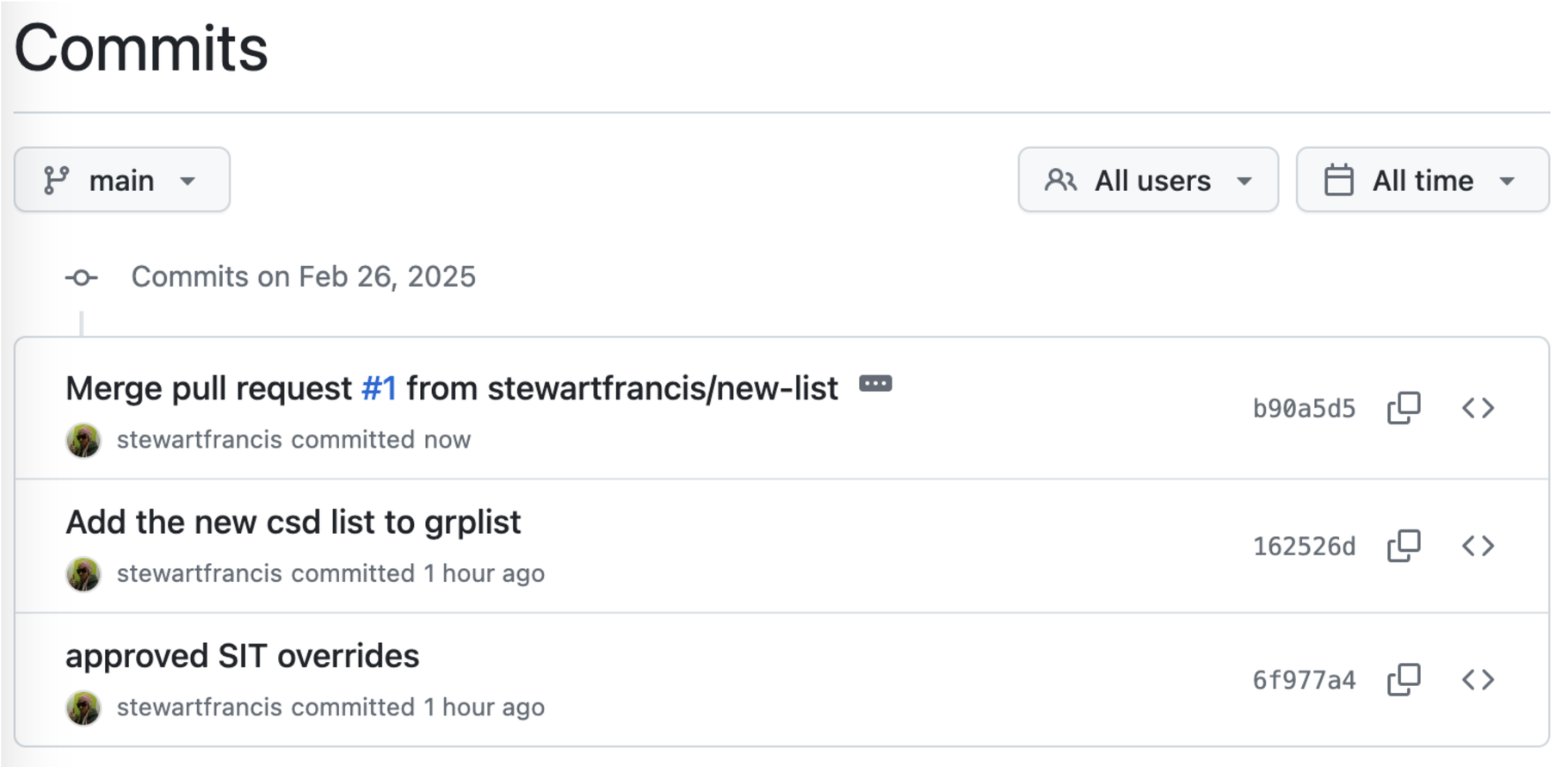
What’s the new process for changing my SIT parameters?

- 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



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



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?

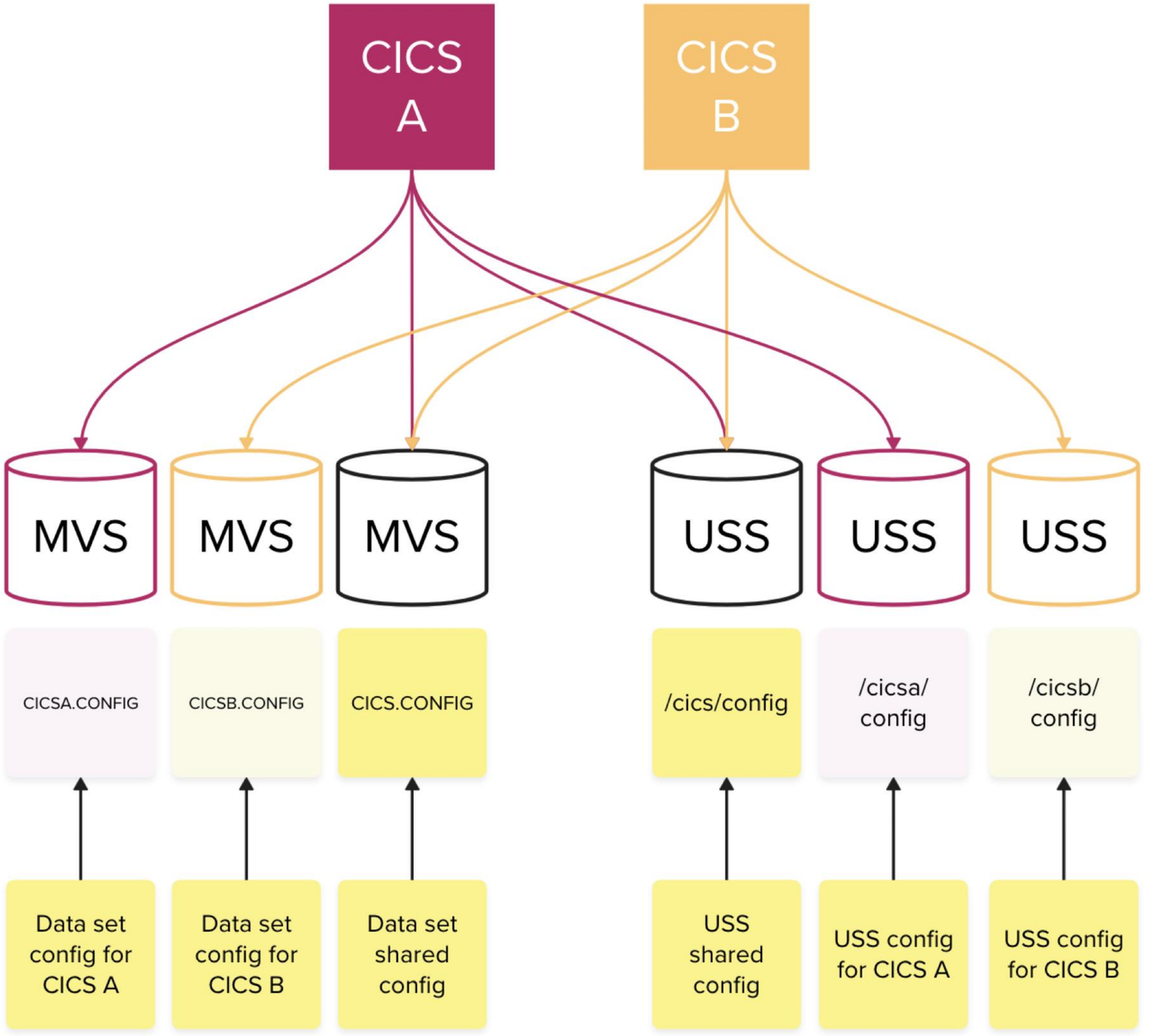
- CICS configuration is **shared**

- 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

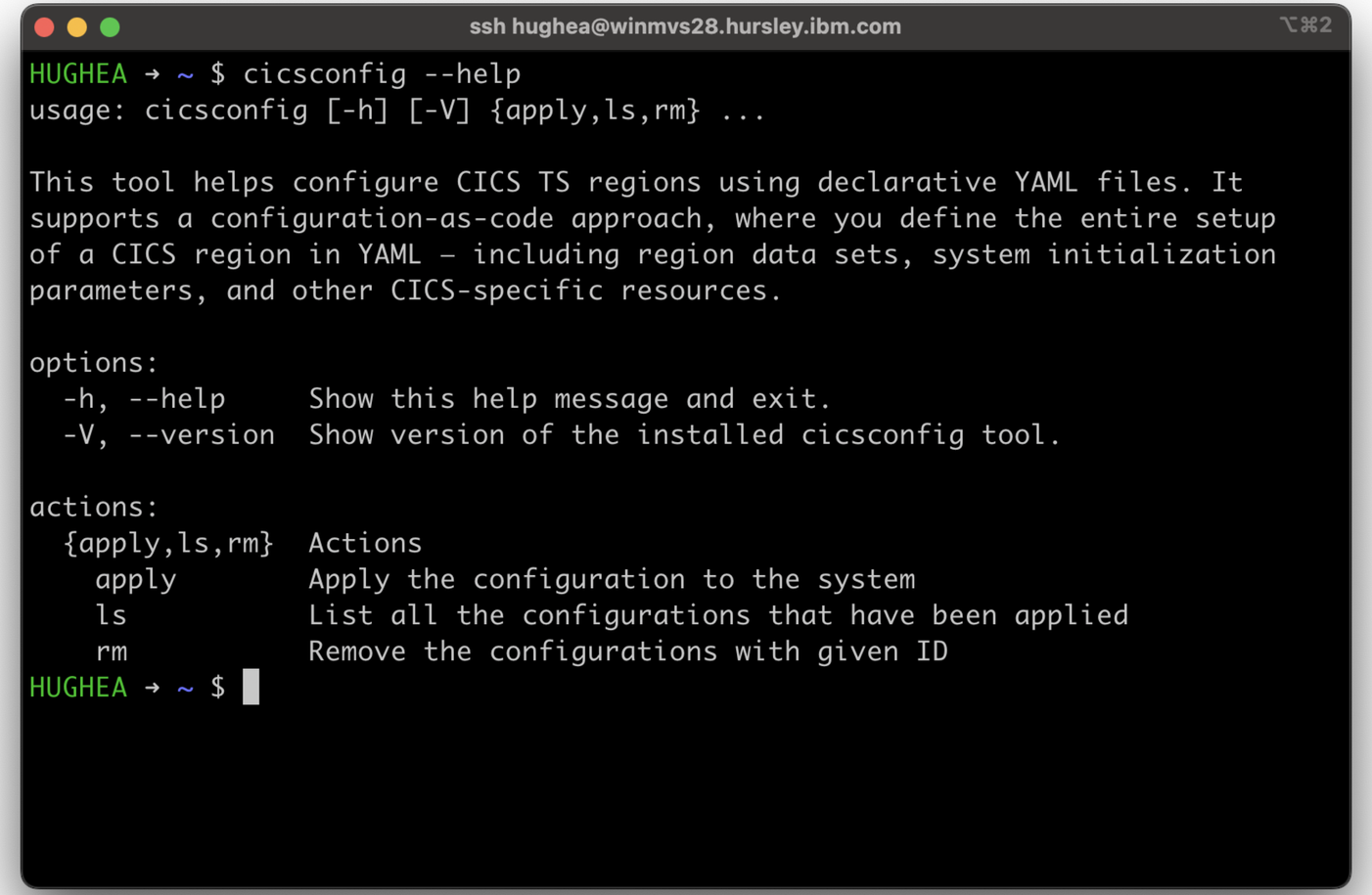


Let’s configure some CICS regions…

CICS TS configuration tool

Translates YAML into target configuration for CICS
- Allocates all the required data sets
- Generates the JCL
- Populates the CSD
- Etc.

No runtime changes – this works with any version of CICS TS



CICS TS resource builder

- A configuration-as-code tool 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
- Application developers can then author definition files, subject to the constraints provided by the system programmer

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
```

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

Mortgages (cics.model.json)

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

cicsconfig support for resource builder

- 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

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

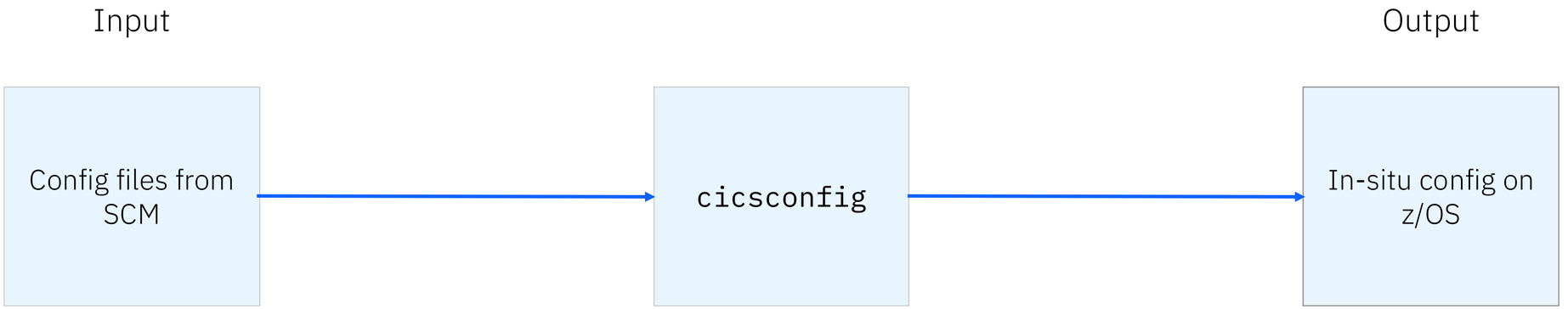
The `cicsconfig` tool has integrated support for resource builder yaml files

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

```yaml
csd:
   content:
     - type: resource_builder
       model: catman.cicsresourcemodel.yaml
       definitions: catman.cicsresourcedefinitions.yaml
```

* But it also supports CSDUP scripts as a way of updating your CSD.

High level architecture

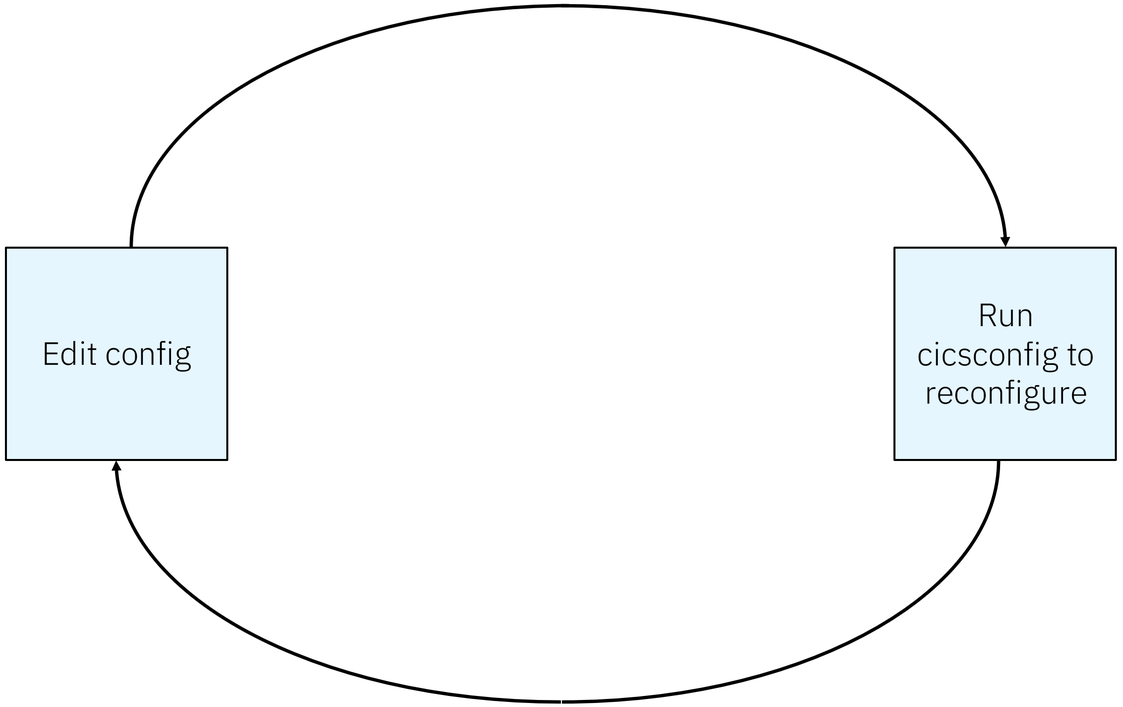


YAML-based config language +
Other assets like:
- Resource definition yaml
- JVM profiles

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 profiles etc

Why is it important that it’s declarative?



- You specify what – not how. The tool knows how to translate YAML into configuration that CICS already understands
- Declarative config is much easier to implement and reason about
- No confusing separation of provision/deprovision
- Deploying any configuration change is the exact same process
- Config files represent the truth of how the system is configured – no configuration drift

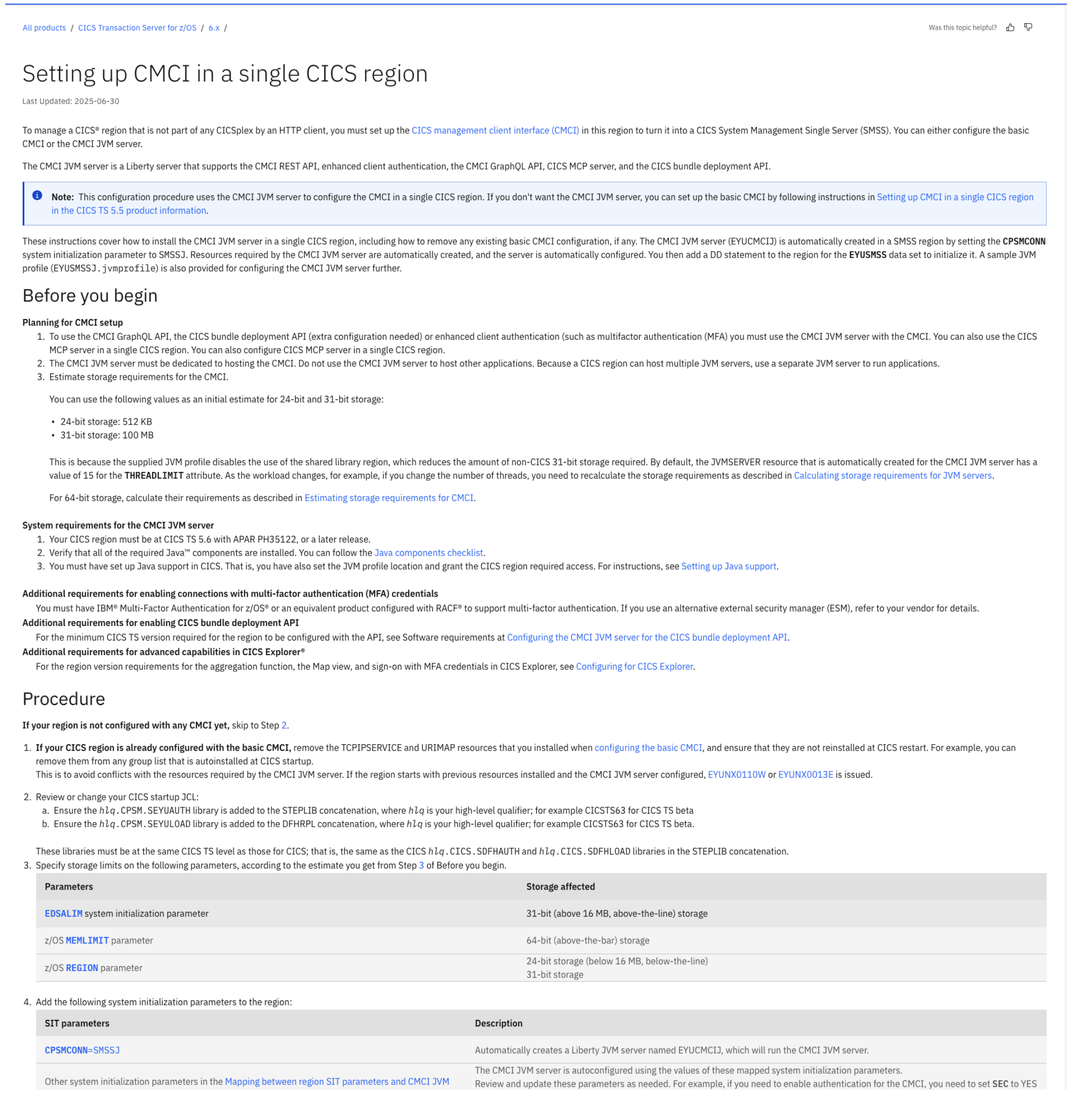
What else can it do?

Configuring CMCI JVM server in our SMSS region

We're going to follow the documentation: https://www.ibm.com/docs/en/cics-ts/6.x?topic=cmci-setting-up-in-single-cics-region

We need to:

- Set the CPSMCONN=SMSSJ SIT parameter
- Add the EYUSMSS DD the JCL to set the generated port, and other parameters
- Create the EYUSMSS.jvmprofile
- Set the JVM profile dir



All products / CICS Transaction Server for z/OS / 6.x /

Was this topic helpful?

Setting up CMCI in a single CICS region

Last Updated: 2025-06-30

To manage a CICS® region that is not part of any CICSplex by an HTTP client, you must set up the CICS management client interface (CMCI) in this region to turn it into a CICS System Management Single Server (SMSS). You can either configure the basic CMCI or the CMCI JVM server.

The CMCI JVM server is a Liberty server that supports the CMCI REST API, enhanced client authentication, the CMCI GraphQL API, CICS MCP server, and the CICS bundle deployment API.

Note: This configuration procedure uses the CMCI JVM server to configure the CMCI in a single CICS region. If you don't want the CMCI JVM server, you can set up the basic CMCI by following instructions in Setting up CMCI in a single CICS region in the CICS TS 5.5 product information.

These instructions cover how to install the CMCI JVM server in a single CICS region, including how to remove any existing basic CMCI configuration, if any. The CMCI JVM server (EYUCMCIJ) is automatically created in a SMSS region by setting the **CPSMCONN** system initialization parameter to SMSSJ. Resources required by the CMCI JVM server are automatically created, and the server is automatically configured. You then add a DD statement to the region for the **EYUSMSS** data set to initialize it. A sample JVM profile (`EYUSMSSJ.jvmprofile`) is also provided for configuring the CMCI JVM server further.

Before you begin

Planning for CMCI setup

1. To use the CMCI GraphQL API, the CICS bundle deployment API (extra configuration needed) or enhanced client authentication (such as multifactor authentication (MFA) you must use the CMCI JVM server with the CMCI. You can also use the CICS MCP server in a single CICS region. You can also configure CICS MCP server in a single CICS region.
2. The CMCI JVM server must be dedicated to hosting the CMCI. Do not use the CMCI JVM server to host other applications. Because a CICS region can host multiple JVM servers, use a separate JVM server to run applications.
3. Estimate storage requirements for the CMCI.

   You can use the following values as an initial estimate for 24-bit and 31-bit storage:

   - 24-bit storage: 512 KB
   - 31-bit storage: 100 MB

   This is because the supplied JVM profile disables the use of the shared library region, which reduces the amount of non-CICS 31-bit storage required. By default, the JVMSERVER resource that is automatically created for the CMCI JVM server has a value of 15 for the **THREADLIMIT** attribute. As the workload changes, for example, if you change the number of threads, you need to recalculate the storage requirements as described in Calculating storage requirements for JVM servers.

   For 64-bit storage, calculate their requirements as described in Estimating storage requirements for CMCI.

System requirements for the CMCI JVM server

1. Your CICS region must be at CICS TS 5.6 with APAR PH35122, or a later release.
2. Verify that all of the required Java™ components are installed. You can follow the Java components checklist.
3. You must have set up Java support in CICS. That is, you have also set the JVM profile location and grant the CICS region required access. For instructions, see Setting up Java support.

Additional requirements for enabling connections with multi-factor authentication (MFA) credentials

You must have IBM® Multi-Factor Authentication for z/OS® or an equivalent product configured with RACF® to support multi-factor authentication. If you use an alternative external security manager (ESM), refer to your vendor for details.

Additional requirements for enabling CICS bundle deployment API

For the minimum CICS TS version required for the region to be configured with the API, see Software requirements at Configuring the CMCI JVM server for the CICS bundle deployment API.

Additional requirements for advanced capabilities in CICS Explorer®

For the region version requirements for the aggregation function, the Map view, and sign-on with MFA credentials in CICS Explorer, see Configuring for CICS Explorer.

Procedure

If your region is not configured with any CMCI yet, skip to Step 2.

1. **If your CICS region is already configured with the basic CMCI,** remove the TCPIPSERVICE and URIMAP resources that you installed when configuring the basic CMCI, and ensure that they are not reinstalled at CICS restart. For example, you can remove them from any group list that is autoinstalled at CICS startup.
   This is to avoid conflicts with the resources required by the CMCI JVM server. If the region starts with previous resources installed and the CMCI JVM server configured, EYUNX0110W or EYUNX0013E is issued.

2. Review or change your CICS startup JCL:
   a. Ensure the *hlq*`.CPSM.SEYUAUTH` library is added to the STEPLIB concatenation, where *hlq* is your high-level qualifier; for example CICSTS63 for CICS TS beta
   b. Ensure the *hlq*`.CPSM.SEYULOAD` library is added to the DFHRPL concatenation, where *hlq* is your high-level qualifier; for example CICSTS63 for CICS TS beta.

   These libraries must be at the same CICS TS level as those for CICS; that is, the same as the CICS *hlq*`.CICS.SDFHAUTH` and *hlq*`.CICS.SDFHLOAD` libraries in the STEPLIB concatenation.
3. Specify storage limits on the following parameters, according to the estimate you get from Step 3 of Before you begin.

Parameters	Storage affected
EDSALIM system initialization parameter	31-bit (above 16 MB, above-the-line) storage
z/OS **MEMLIMIT** parameter	64-bit (above-the-bar) storage
z/OS **REGION** parameter	24-bit storage (below 16 MB, below-the-line) 31-bit storage

4. Add the following system initialization parameters to the region:

SIT parameters	Description
CPSMCONN=SMSSJ	Automatically creates a Liberty JVM server named EYUCMCIJ, which will run the CMCI JVM server.
Other system initialization parameters in the Mapping between region SIT parameters and CMCI JVM	The CMCI JVM server is autoconfigured using the values of these mapped system initialization parameters. Review and update these parameters as needed. For example, if you need to enable authentication for the CMCI, you need to set **SEC** to YES

Let’s add some Java…

Variables

Variables borrow Ansible syntax. Unlike Ansible, you can’t use arbitrary Jinja expressions

Variables can be provided on the CLI or specified in a vars block in the YAML

Variables can be used to construct the value of any property

The value of any property can be used as a variable too!

Order in the document doesn’t matter!

```
cicsconfig apply region.yaml

vars:
    sysid: ZOEA
  cics_region:
    sysid: ”{{ vars.sysid }}”
    applid: “{{ cics_region.applid }}”
    .
    .
    .
```

```
cicsconfig apply -e sysid=ZOEA
  region.yaml

cics_region:
    sysid: ”{{ vars.sysid }}”
    applid: “{{ cics_region.applid }}”
    .
    .
    .
```

Extensions

What if there was an even simpler way to configure CMCI in a CICS region, that felt more native?

Cicsconfig YAML supports an extensions property which we can use as a mechanism to implement that

Extensions can programmatically contribute additional configuration, that’s not built into the underlying YAML document, at the time the configuration is applied

Our intention is to open up the extension python API to clients, and vendors

```
cics_region:

    ...

    extensions:
      cics_cmci:
        cmci_port: 12345
        ssl: “YES”
        cmci_auth: BASIC
```

TODO list:

- Add EYUSMSS DD
- Populate with config
- Set SIT parameter CPSMCONN to SMSSJ

JVM profiles

Top level configuration option for JVM profile directory

This is used to set the JVMPROFILEDIR sit parameter in the JCL

JVM profiles can be specified in-line, or alternatively external files can be referenced

However they’re specified, profiles are copied into the target jvm_profile_dir so they’ll be available to the resulting region

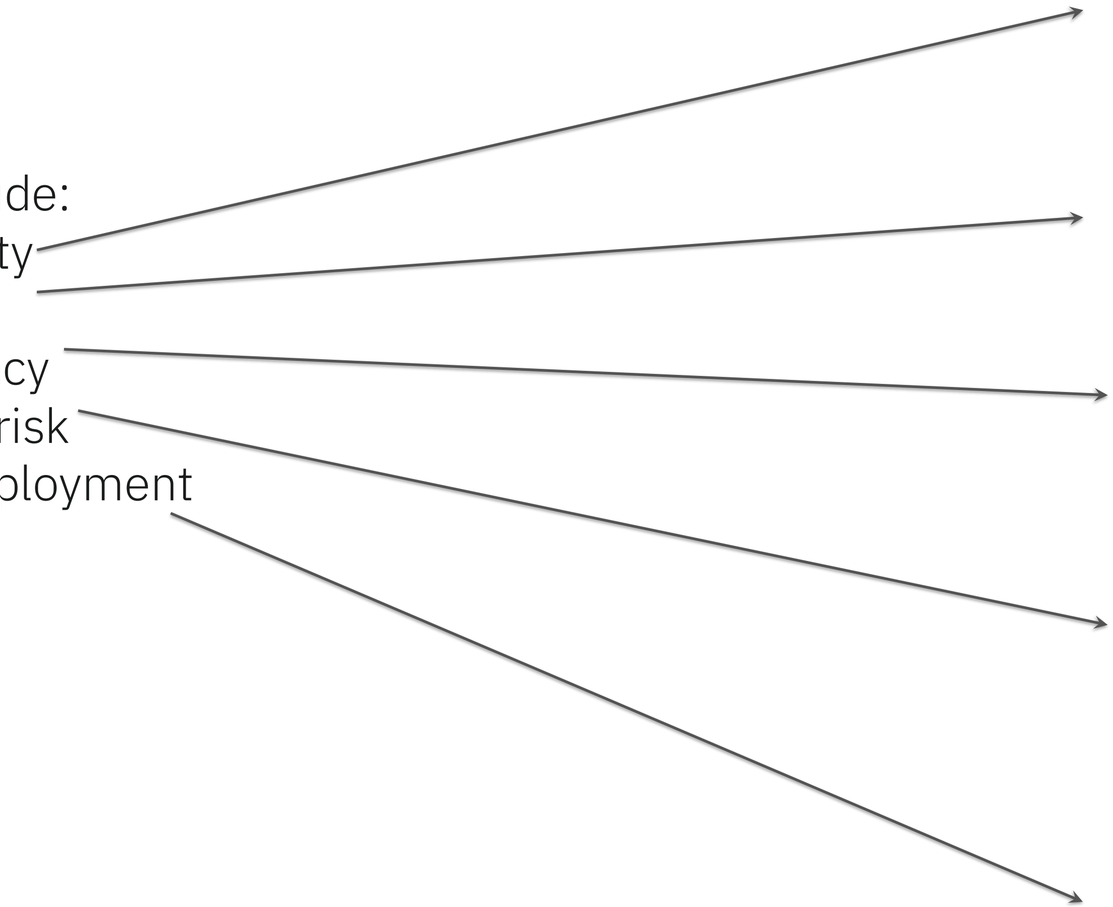
```
jvm_profile_dir: /u/hughea/{{ vars.applid }}/jvmprofiles

  jvm_profiles:
    - name: EYUSMSSJ
      existing: true
      source_path: /u/hughea/jvmprofiles

    - name: OTHER
      existing: false
      properties: |
        WORK_DIR=.
        -Xms128M
        -Xmx1G
        -Xmso1M
        -Dfile.encoding=ISO-8859-1
        ...
```

Why should you care?

Advantages

Benefits include:

- Auditability
- Back-out
- Consistency
- Reduced risk
- Faster deployment



Every change (e.g. JCL, CSD resources, etc) is associate with a reviewed SCM commit

Undo the last commit

Environments built from the same SCM commit are de-facto consistent

All deployments follow the same automated process, no manual intervention

Any changes can be deployed in the same way, with an SCM commit

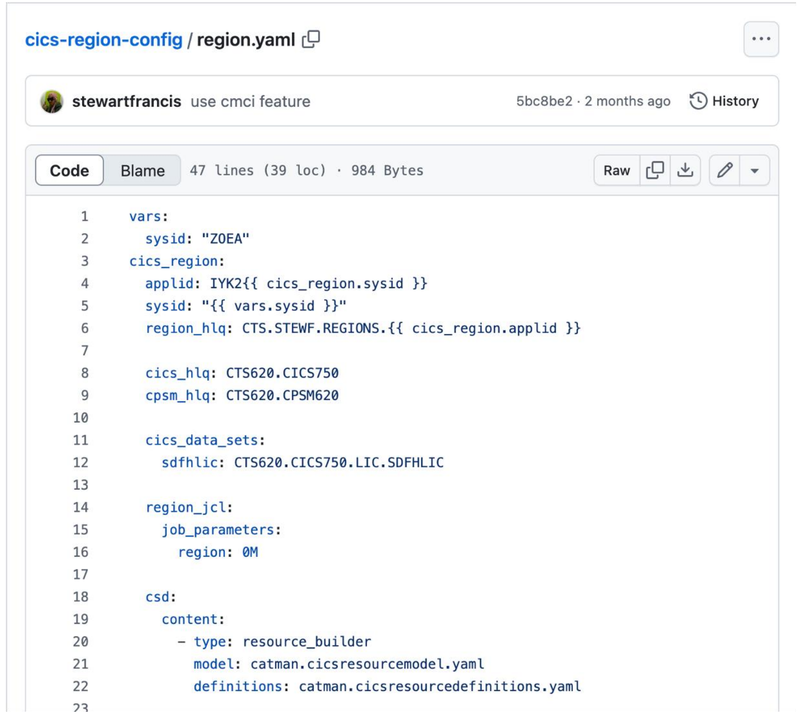
Auditable process

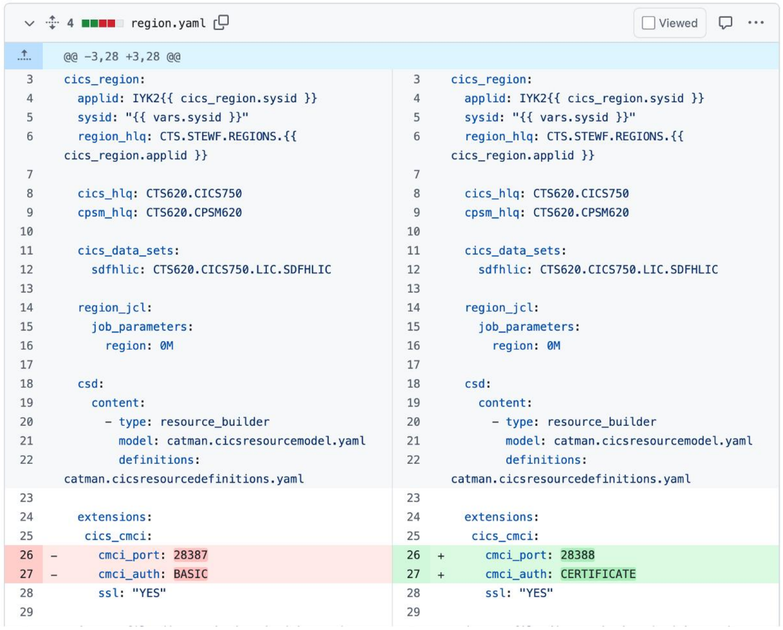
Configuration is stored and edited into an SCM.

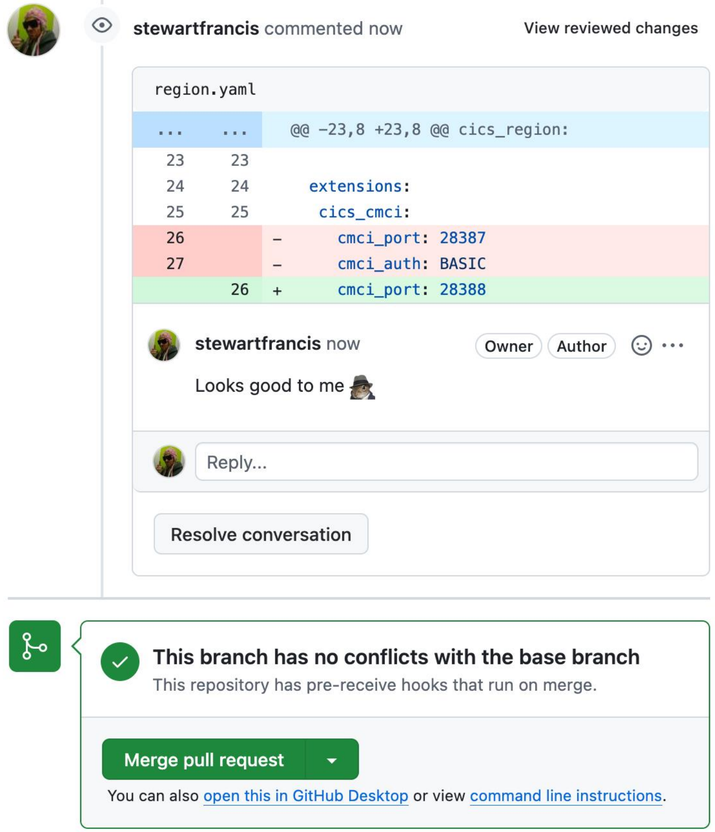
Changes are made to the configuration, and a pull request has been created.

Comments have been left on the pull request.

Identified reviewer will approve or reject the request with history attached for future reference.







How does this simplify the configuration?

Unified model for ‘all’ CICS configuration

Standardized configuration into a single format that is incredibly popular in the industry

Powerful editing support with:
- Code-completion (type-ahead suggestions)
- Integrated documentation
- Validation

Introduced higher-level configuration semantics:
- Naming conventions
- Sensible defaults
- Extensions

But also….

Standardized deployment mechanism for ‘any’ configuration change – e.g. could be automated in a pipeline with Ansible.

Workflow based on technologies that are common in the industry.

Let’s automate this!

Red Hat® Ansible® Certified Content for IBM Z®: Collections

Available on Ansible Galaxy here and on Automation Hub			
IBM z/OS® Core		ibm_zos_core Provided by ibm 23 Modules 0 Roles 26 Plugins 0 Dependencies mvs z data_set 9 more	401,000 Downloads
IBM z/OS® CICS®		ibm_zos_cics Provided by ibm 15 Modules 0 Roles 35 Plugins 0 Dependencies z zos cics 4 more	23,000 Downloads
IBM z/OS® IMS™		ibm_zos_ims Provided by ibm 8 Modules 0 Roles 9 Plugins 1 Dependency mvs zos_ims z 11 more	54,000 Downloads
IBM z/OS® Management Facility (z/OSMF)		ibm_zosmf Provided by ibm 3 Modules 13 Roles 4 Plugins 0 Dependencies mvs liberty_provisioning_template z	13,500 Downloads
IBM Z® Hardware Management Console		ibm_zhmc Provided by ibm 28 Modules 0 Roles 1 Plugin 0 Dependencies z hmc ibm 2 more	114,000 Downloads
IBM Z® System Automation		ibm_zos_sysauto Provided by ibm 0 Modules 2 Roles 0 Plugins 0 Dependencies z zos ibm 6 more	4,100 Downloads

Collectively

- 83 Modules
- 15 Roles

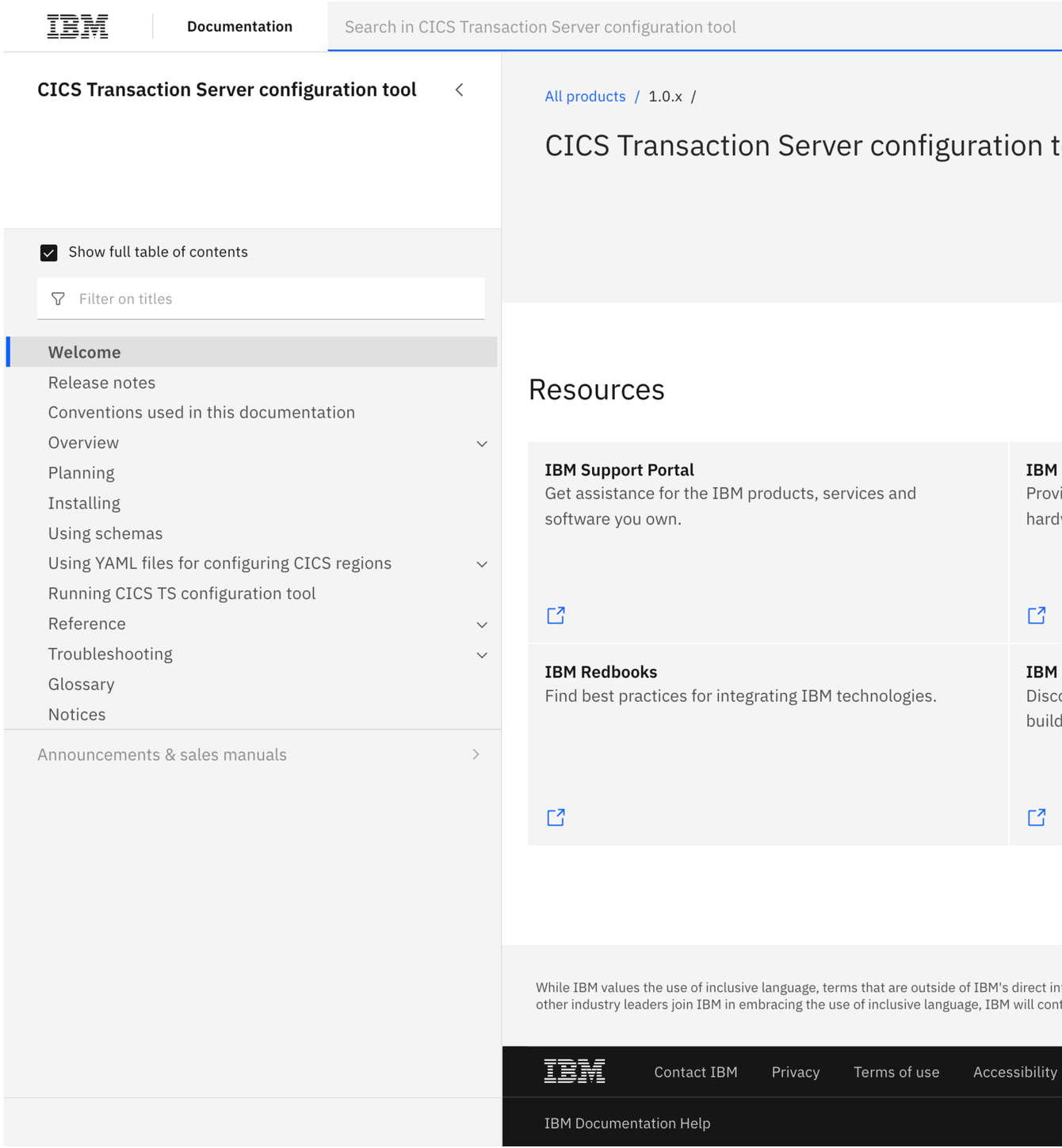
Ready to try it out?

CICS TS configuration tool is available on the CICS TS Ruby Beta site:

ibm.biz/cicsconfig-beta

Documentation available online:

ibm.biz/cicsconfig-docs



Questions