IBM SPS/1PL DevSecOps Extensions for Concert

Rong Chang
rong@us.ibm.com

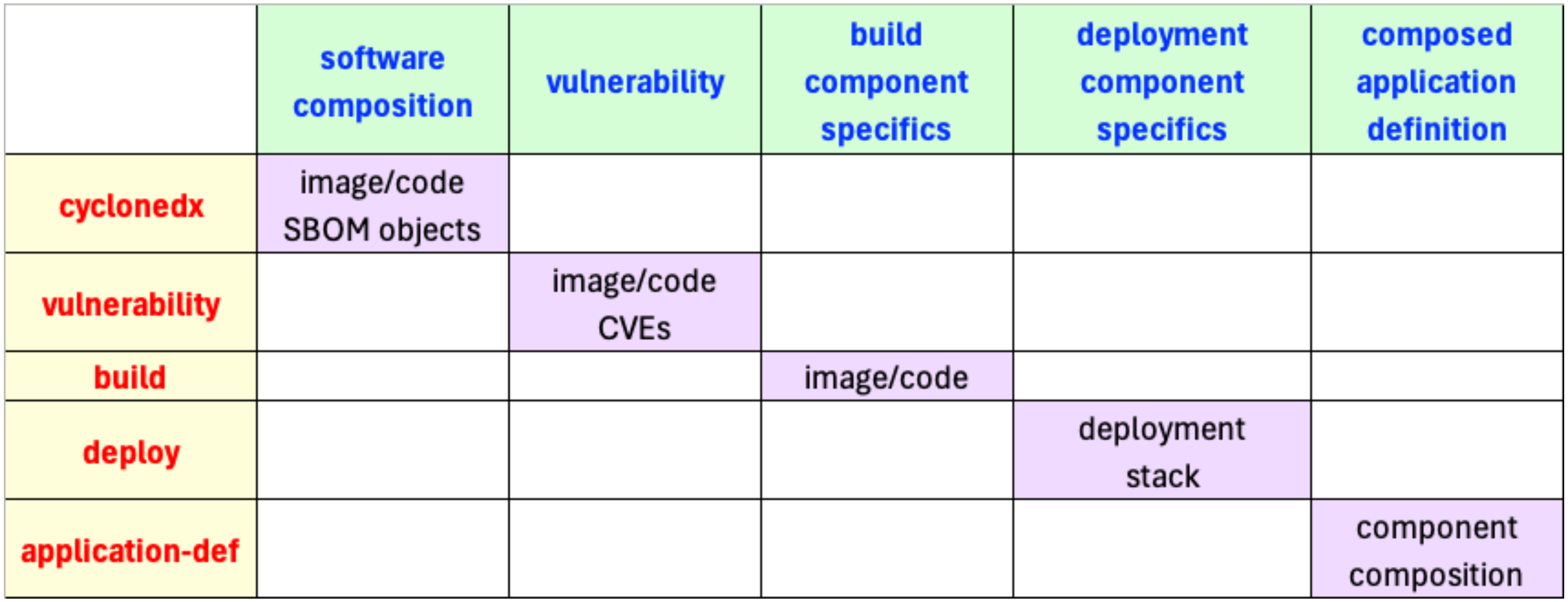
https://github.ibm.com/roja/concert-utils/tree/main/utils-sps1pl_for_concert

September 20, 2024

Outline

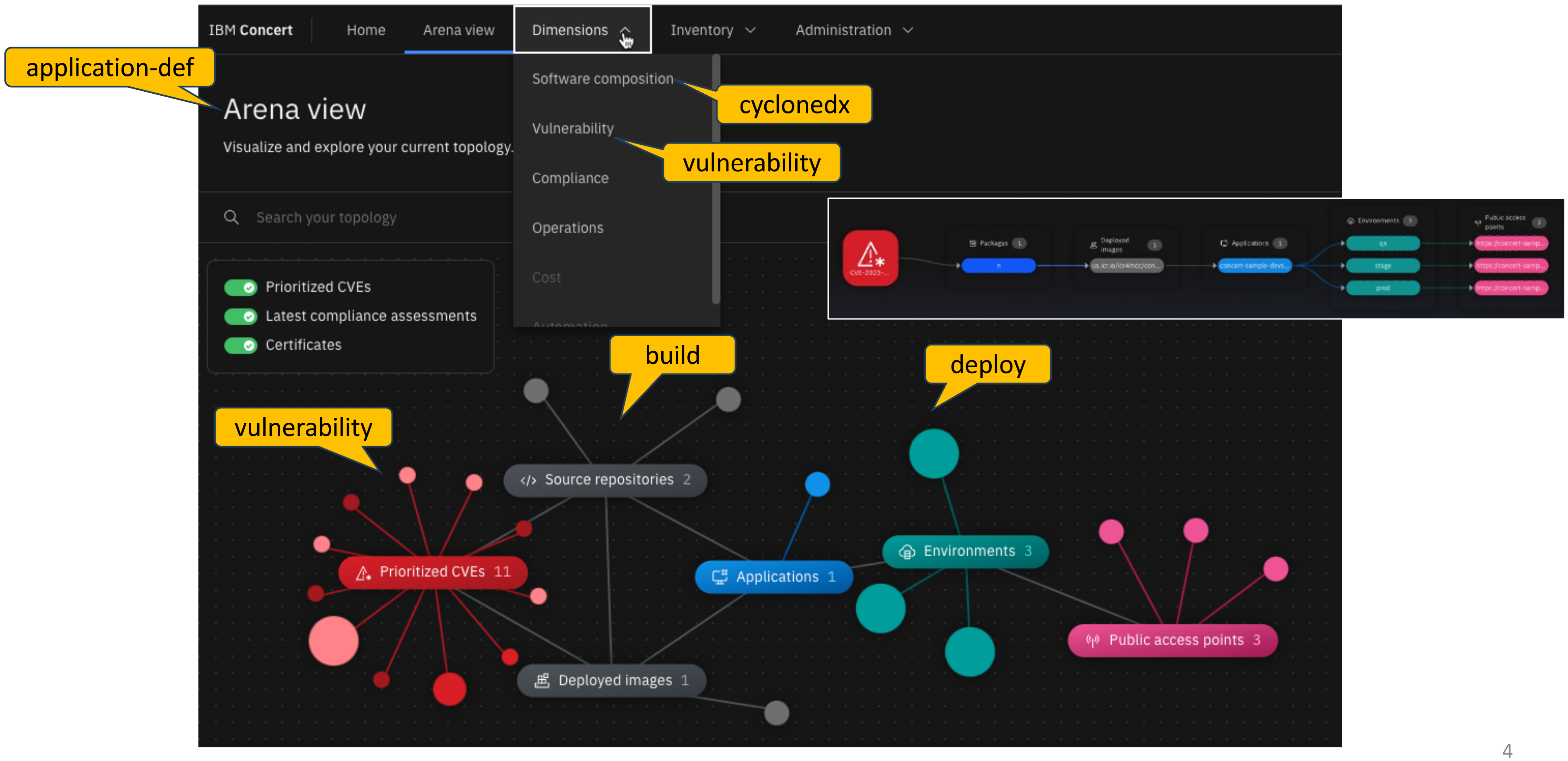
- Concert Input File Types
- Pipeline Extension Scripts for Concert
- DevSecOps Implementation Framework of IBM SPS/1PL

Concert Input File Types

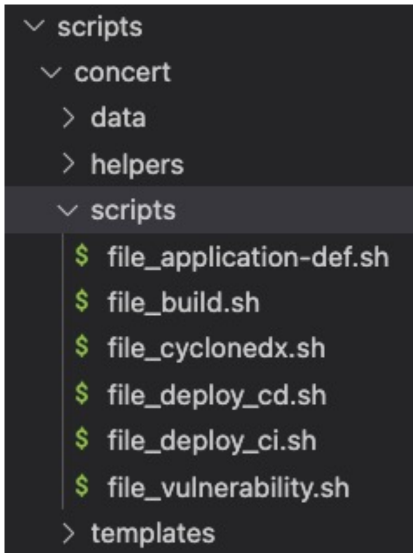


	software composition	vulnerability	build component specifics	deployment component specifics	composed application definition
cyclonedx	image/code SBOM objects				
vulnerability		image/code CVEs			
build			image/code		
deploy				deployment stack	
application-def					component composition

Sample Screenshot of Concert Console



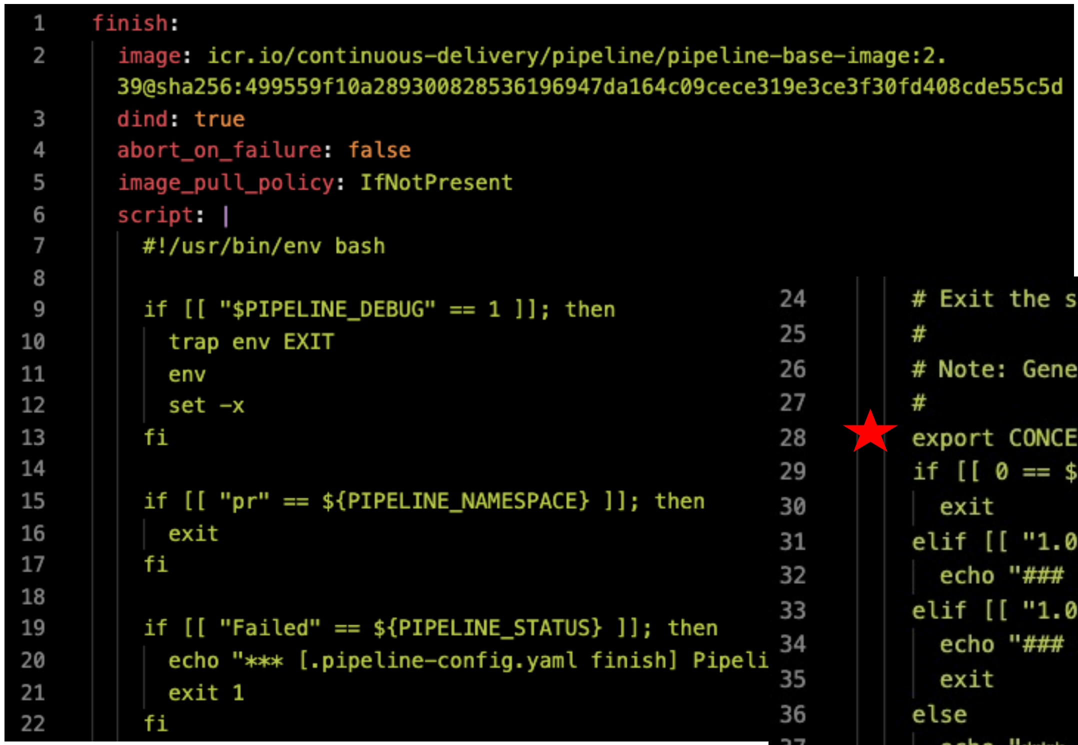
Extending Existing SPS/1PL Pipelines for Concert



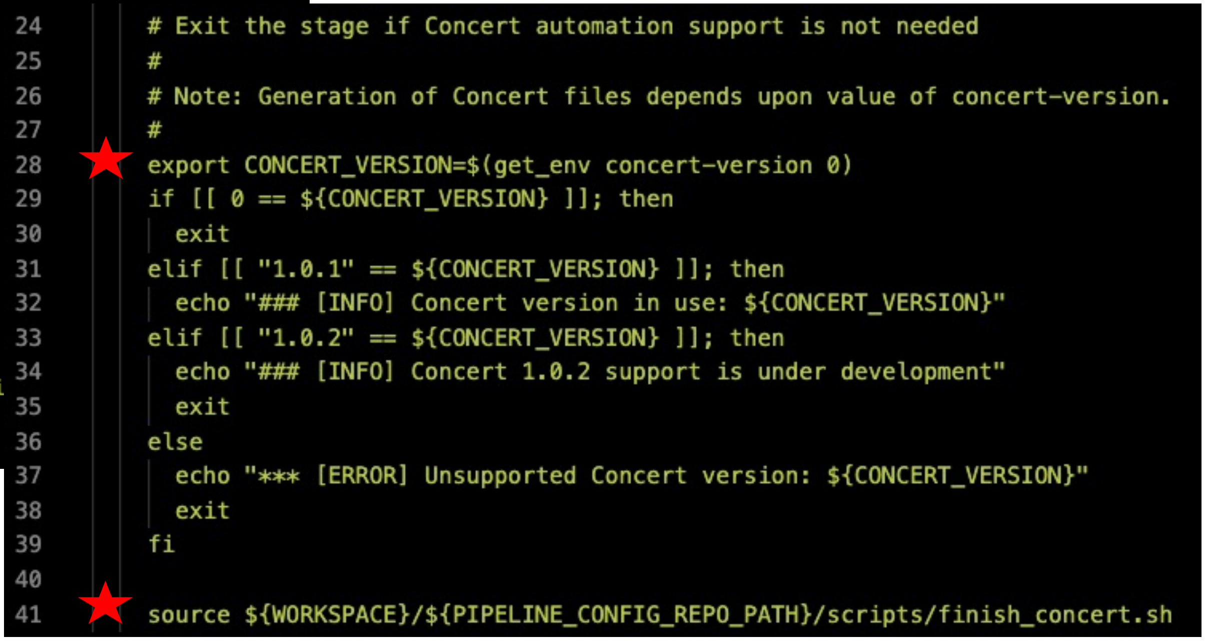
```
scripts
  concert
    data
    helpers
    scripts
      file_application-def.sh
      file_build.sh
      file_cyclonedx.sh
      file_deploy_cd.sh
      file_deploy_ci.sh
      file_vulnerability.sh
    templates
```

- Execution sequence when common SPS/1PL repo file structure is used: [**./.pipeline-config.yaml**] -> [**./scripts/finish_concert.sh**] -> scripts in [**./scripts/concert/scripts/**]
- [./utils-sps1pl_for_concert/scripts/sample_pipeline-config.yaml]
  - Sample "finish" script in common SPS/1PL configure file [**./.pipeline-config.yaml**]
- [./utils-sps1pl_for_concert/scripts/concert/sample_finish_concert.sh]
  - Sample [**./scripts/finish_concert.sh**] that sets additional Concert-specific environment variables to those configured in the pipeline settings (e.g., secure value of Concert service API key).
- [./utils-sps1pl_for_concert/]: Directory root for IBM SPS/1PL DevSecOps extension scripts for Concert
  - **scripts**: Common "scripts" directory of an IBM SPS/1PL pipeline
    - **concert**: Directory root for Concert-specific scripts and template files
      - **data**: Data exchange directory between pipelinerun and Concert Toolkit image container
      - **helpers**: Wrapper scripts for the Concert Toolkit image container in use (source: [./helpers])
      - **scripts**: Pipeline extension scripts for automated Concert file generation and upload. There is one script for each type of Concert input files except two scripts are used for generating & upload ConcertDef "deploy" inventory SBOMs, because CI and CD pipelines have different image deployment requirements.
      - **templates**: YAML and/or JSON formatted Concert file generation templates

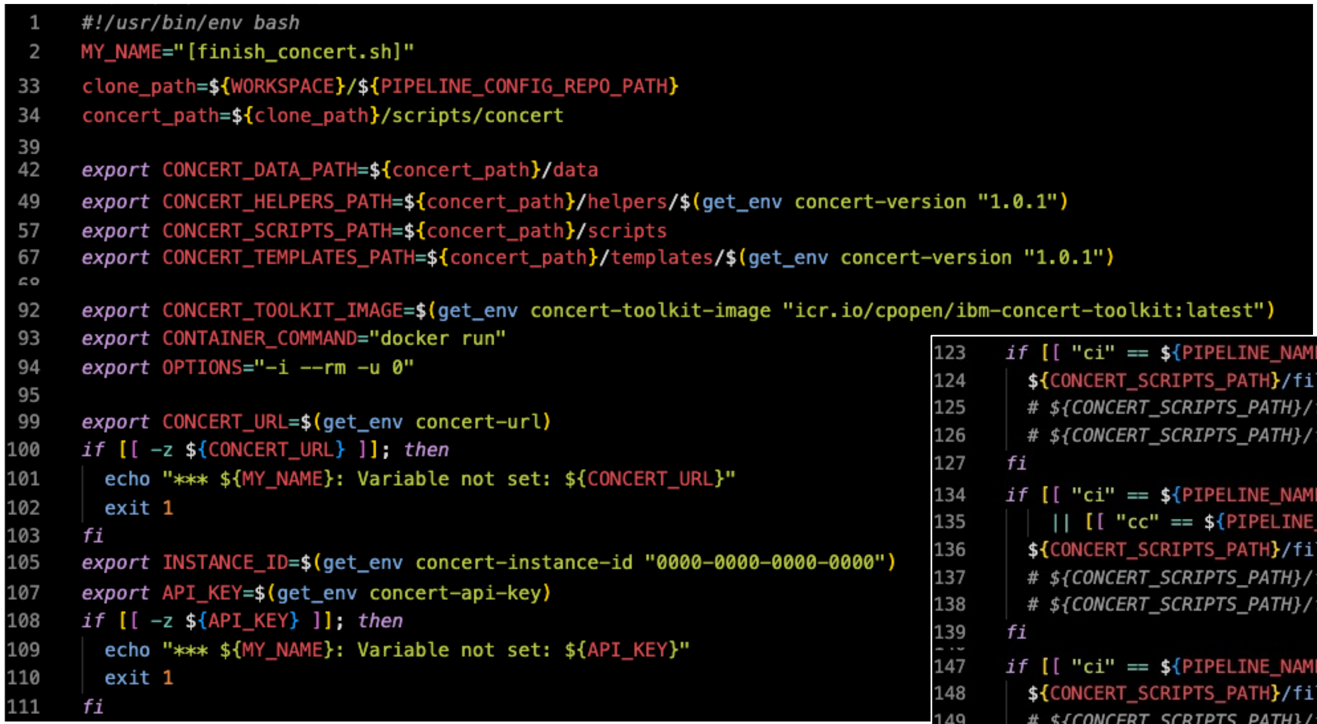
“finish” stage in [./.pipeline-config.yaml]



```
finish:
  image: icr.io/continuous-delivery/pipeline/pipeline-base-image:2.39@sha256:499559f10a289300828536196947da164c09cece319e3ce3f30fd408cde55c5d
  dind: true
  abort_on_failure: false
  image_pull_policy: IfNotPresent
  script: |
    #!/usr/bin/env bash

    if [[ "$PIPELINE_DEBUG" == 1 ]]; then
      trap env EXIT
      env
      set -x
    fi

    if [[ "pr" == ${PIPELINE_NAMESPACE} ]]; then
      exit
    fi

    if [[ "Failed" == ${PIPELINE_STATUS} ]]; then
      echo "*** [.pipeline-config.yaml finish] Pipeli
      exit 1
    fi
```

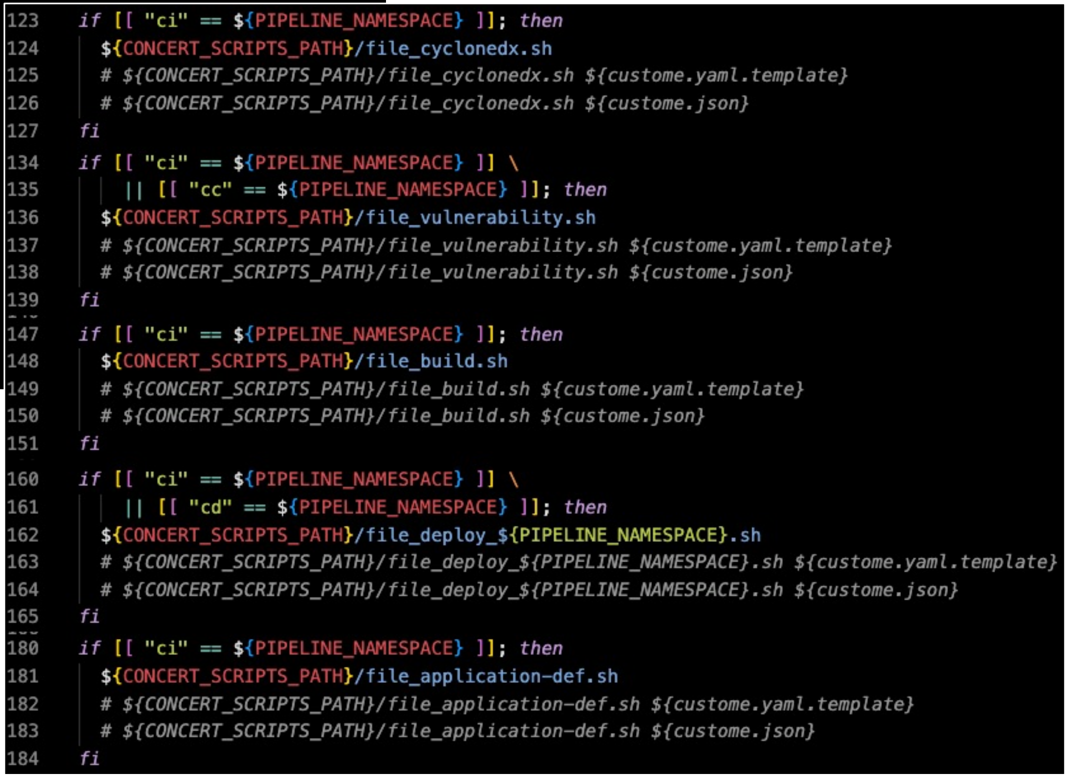


```
    # Exit the stage if Concert automation support is not needed
    #
    # Note: Generation of Concert files depends upon value of concert-version.
    #
    export CONCERT_VERSION=$(get_env concert-version 0)
    if [[ 0 == ${CONCERT_VERSION} ]]; then
      exit
    elif [[ "1.0.1" == ${CONCERT_VERSION} ]]; then
      echo "### [INFO] Concert version in use: ${CONCERT_VERSION}"
    elif [[ "1.0.2" == ${CONCERT_VERSION} ]]; then
      echo "### [INFO] Concert 1.0.2 support is under development"
      exit
    else
      echo "*** [ERROR] Unsupported Concert version: ${CONCERT_VERSION}"
      exit
    fi

    source ${WORKSPACE}/${PIPELINE_CONFIG_REPO_PATH}/scripts/finish_concert.sh
```



```bash
#!/usr/bin/env bash
MY_NAME="[finish_concert.sh]"
clone_path=${WORKSPACE}/${PIPELINE_CONFIG_REPO_PATH}
concert_path=${clone_path}/scripts/concert

export CONCERT_DATA_PATH=${concert_path}/data
export CONCERT_HELPERS_PATH=${concert_path}/helpers/$(get_env concert-version "1.0.1")
export CONCERT_SCRIPTS_PATH=${concert_path}/scripts
export CONCERT_TEMPLATES_PATH=${concert_path}/templates/$(get_env concert-version "1.0.1")

export CONCERT_TOOLKIT_IMAGE=$(get_env concert-toolkit-image "icr.io/cpopen/ibm-concert-toolkit:latest")
export CONTAINER_COMMAND="docker run"
export OPTIONS="-i --rm -u 0"

export CONCERT_URL=$(get_env concert-url)
if [[ -z ${CONCERT_URL} ]]; then
  echo "*** ${MY_NAME}: Variable not set: ${CONCERT_URL}"
  exit 1
fi
export INSTANCE_ID=$(get_env concert-instance-id "0000-0000-0000-0000")
export API_KEY=$(get_env concert-api-key)
if [[ -z ${API_KEY} ]]; then
  echo "*** ${MY_NAME}: Variable not set: ${API_KEY}"
  exit 1
fi
```

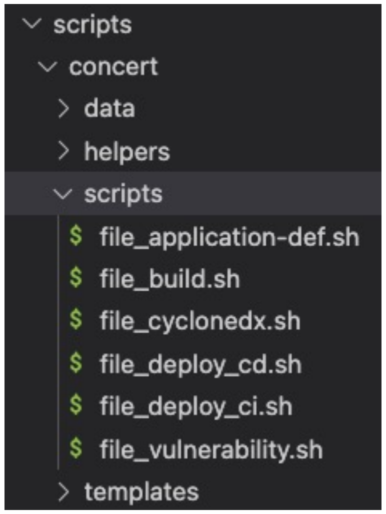
Sample [./scripts/finish_concert.sh]



```bash
if [[ "ci" == ${PIPELINE_NAMESPACE} ]]; then
  ${CONCERT_SCRIPTS_PATH}/file_cyclonedx.sh
  # ${CONCERT_SCRIPTS_PATH}/file_cyclonedx.sh ${custome.yaml.template}
  # ${CONCERT_SCRIPTS_PATH}/file_cyclonedx.sh ${custome.json}
fi
if [[ "ci" == ${PIPELINE_NAMESPACE} ]] \
    || [[ "cc" == ${PIPELINE_NAMESPACE} ]]; then
  ${CONCERT_SCRIPTS_PATH}/file_vulnerability.sh
  # ${CONCERT_SCRIPTS_PATH}/file_vulnerability.sh ${custome.yaml.template}
  # ${CONCERT_SCRIPTS_PATH}/file_vulnerability.sh ${custome.json}
fi
if [[ "ci" == ${PIPELINE_NAMESPACE} ]]; then
  ${CONCERT_SCRIPTS_PATH}/file_build.sh
  # ${CONCERT_SCRIPTS_PATH}/file_build.sh ${custome.yaml.template}
  # ${CONCERT_SCRIPTS_PATH}/file_build.sh ${custome.json}
fi
if [[ "ci" == ${PIPELINE_NAMESPACE} ]] \
    || [[ "cd" == ${PIPELINE_NAMESPACE} ]]; then
  ${CONCERT_SCRIPTS_PATH}/file_deploy_${PIPELINE_NAMESPACE}.sh
  # ${CONCERT_SCRIPTS_PATH}/file_deploy_${PIPELINE_NAMESPACE}.sh ${custome.yaml.template}
  # ${CONCERT_SCRIPTS_PATH}/file_deploy_${PIPELINE_NAMESPACE}.sh ${custome.json}
fi
if [[ "ci" == ${PIPELINE_NAMESPACE} ]]; then
  ${CONCERT_SCRIPTS_PATH}/file_application-def.sh
  # ${CONCERT_SCRIPTS_PATH}/file_application-def.sh ${custome.yaml.template}
  # ${CONCERT_SCRIPTS_PATH}/file_application-def.sh ${custome.json}
fi
```

Structure of SPS/1PL Pipeline Extension Scripts for Concert

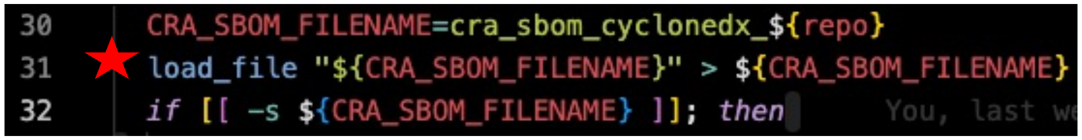
- Each script in [concert/scripts/] generates & uploads a specific type of Concert files.
  - Sequence of execution (as sub-shell): **cyclonedx** -> **vulnerability** -> **build** -> **deploy** -> **application-def**
- Common runtime environment for all scripts
  - SPS/1PL framework implementation, including repos, CLIs, and environment variables
  - A docker runtime (via DinD) that enables executing a Concert Toolkit image by "docker run"
    - **icr.io/cpopen/ibm-concert-toolkit:latest**
  - Environment variables (including pipeline settings) compiled by the invoking script [finish_concert.sh]
- Common script structure for **ConcertDef** file generation and upload
  1. Assure in-scope pipeline namespace
  2. Set file path for the JSON/YAML-formatted Concert template in use
  3. Process in-scope image inventory entries in sequence
  4. Generate JSON-formatted **ConcertDef** file (**build**, **deploy**, or **application-def**)
  5. Conditionally upload generated file to a COS/S3 bucket for local use
  6. Upload generated file to Concert
  - Note: Two deploy scripts: one for CI deployment and the other for CD deployment



```
scripts
  concert
    data
    helpers
    scripts
      $ file_application-def.sh
      $ file_build.sh
      $ file_cyclonedx.sh
      $ file_deploy_cd.sh
      $ file_deploy_ci.sh
      $ file_vulnerability.sh
    templates
```

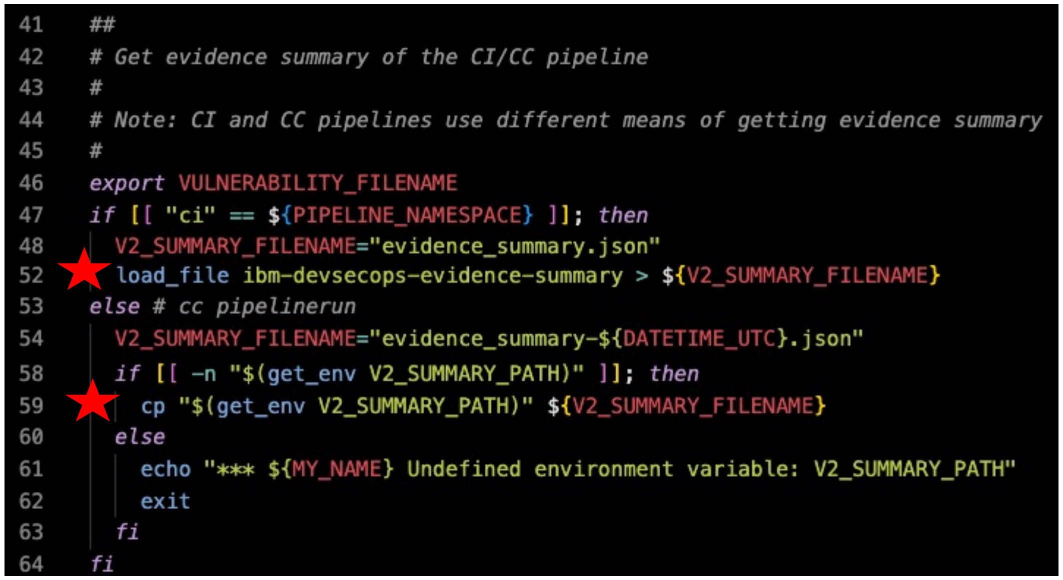
Generation of **cyclonedx** & **vulnerability** Files

- CycloneDX SBOM files are currently generated by SPS/1PL CRA



```
CRA_SBOM_FILENAME=cra_sbom_cyclonedx_${repo}
load_file "${CRA_SBOM_FILENAME}" > ${CRA_SBOM_FILENAME}
if [[ -s ${CRA_SBOM_FILENAME} ]]; then      You, last we
```

- CVEs for a CSV-formatted Concert vulnerability are currently generated via SPS/1PL “evidence summary”.



```
##
# Get evidence summary of the CI/CC pipeline
#
# Note: CI and CC pipelines use different means of getting evidence summary
#
export VULNERABILITY_FILENAME
if [[ "ci" == ${PIPELINE_NAMESPACE} ]]; then
  V2_SUMMARY_FILENAME="evidence_summary.json"
  load_file ibm-devsecops-evidence-summary > ${V2_SUMMARY_FILENAME}
else # cc pipelinerun
  V2_SUMMARY_FILENAME="evidence_summary-${DATETIME_UTC}.json"
  if [[ -n "$(get_env V2_SUMMARY_PATH)" ]]; then
    cp "$(get_env V2_SUMMARY_PATH)" ${V2_SUMMARY_FILENAME}
  else
    echo "*** ${MY_NAME} Undefined environment variable: V2_SUMMARY_PATH"
    exit
  fi
fi
```

Sample Contents of an SPS/1PL Image Inventory Entry (created by CI Pipelineruns)

Image URI

Code Repo URL

Sha of Code Repo Commit

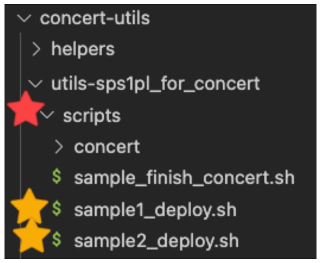
```
{
  "version": "f4547ea773f1831461a3f09e7a1b1976fa6fe43b",
  "artifact": "us.icr.io/icr4mcc/mern-node-app:20231103182056-main-
f4547ea773f1831461a3f09e7a1b1976fa6fe43b@sha256:3686d54f7e02912cab7f6f678c8af12046bb95feb365a931d8dc04c58e7e75b3",
  "name": "mern-node-app",
  "repository_url": "https://github.ibm.com/rong/mern-node-app",
  "build_number": "58",
  "commit_sha": "f4547ea773f1831461a3f09e7a1b1976fa6fe43b",
  "pipeline_run_id": "b413e91d-cf8c-4f51-aa67-e6a4ebf66ad7",
  "app_artifacts": {
      "app": "mern-node-app",
      "tags": "20231103182056-main-f4547ea773f1831461a3f09e7a1b1976fa6fe43b"
  },
★ "type": "image",
  "sha256": "sha256:3686d54f7e02912cab7f6f678c8af12046bb95feb365a931d8dc04c58e7e75b3",
  "provenance": "us.icr.io/icr4mcc/mern-node-app:20231103182056-main-
f4547ea773f1831461a3f09e7a1b1976fa6fe43b@sha256:3686d54f7e02912cab7f6f678c8af12046bb95feb365a931d8dc04c58e7e75b3",
  "signature":
"owGbwMvMwMHI3pzp+lNlvjnj6QOLkhhSXa12VCslF2WWZCYn5ihZVStlpqTmlWSWVILYKfnJ2alFukWpaalFqXnJqUpWSqXFepnJRXqZ+fpAyiQ3OVk/N7U
oTzcvPyVVN7GgwMrIwMjY0NDA2NDCyMDUTDc3MTNPN83E1MQ8NdHc3DjN0MLY0MTMMNE4zcAy1TzRMMnQ0twsLdEsLdXEOEmpVkcpMzcxPRXJ7t
zEvMy01OlS3ZTMdCAFdEFxRqKRqZmVsZmFWYqpSZp5qoGRpaFRcmKSeZpZmpm5RbJFYpqhkYGJWVKSpWlaapKxmWmipbFhikVKsoFJsqlFqnmquWmS
MciyksoCkJ8SS/JzM5MVkvPzSoDOTS1SKM5Mz0ssKS1KBSnKLyjJzM+DBE5yUSpQcRFCj6mekYmeoRLQqMxcoPMScwuUrAzNLC0NjE0tjI1qazuZjFkYGDkY
ZMUUWapONRr9d/kRsEB1mi0sFliZQDHAwMUpABORVBFg6Ala2/Lt15Tp4mp5XU4/fKPirJV+mNxffvnG0Z3W5hfzpx84K5cSc+jNtwrt1M2P7DgmfVh15F
7NqZZW54+rLr8WnZeyvWSZ9zmLDZJ5enu44u/6PJjwufXPnvaFzxYzOKp+0+9iOXbhZ53PvCqPazUqn049jLgcHP17R53IuqsXn9xMN1x9LXzSlp9/paQThHO
F9v258nvuzMAXa30usYbHWb9b9G6F+xc7Jz23GIXyw56cRr8+pb5J2fTvo1DZ0dxDB3bGFquoPLQqYLQ6yT4jjfVjQehS8SkX+m/5dL/ylQr3vVGl65xTw1M+
pa5urcCqY9oPPevFEr8oCmu33OsUUW+t/Ci65fy254ZP2R5Pcyl0Uec8LWc2YdFz226xr2+5b+uv5dJZvtP2xMf6zz2P6u6cF/i6UM7zirGBwyJXJZlpmy1FWva
vuOzquj98ouAZjix2bl8TnUlT+maWhabszrpZO7X9EsOSKkGBwydd/RqMD7VVz9knaHrn1aadFazaBSkxj9Oq3Z8pxvfq69hFXe1KlOS7Pq+UZ9Od38/zdF99N
8gL/Pa8ffW03Qy9c6XOTipwPfjk49W1leG3AucWiN7Zud/nzOpDO/48+JC2w3n1opk511IqJ/ituNTAdK5/7lxJZqeQVr+tK28eqjI+2cX5pmxl0dRZzOLTKhOvL
J5nu2fWyb8fgjzul0wuXzrxr4nGhIe3DU7PXF0FAA=="
}
```

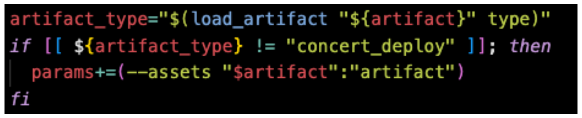
The image inventory entry includes sufficient data for creating a ConcertDef “build” inventory SBOM

Sample concert_deploy artifact creation scripts

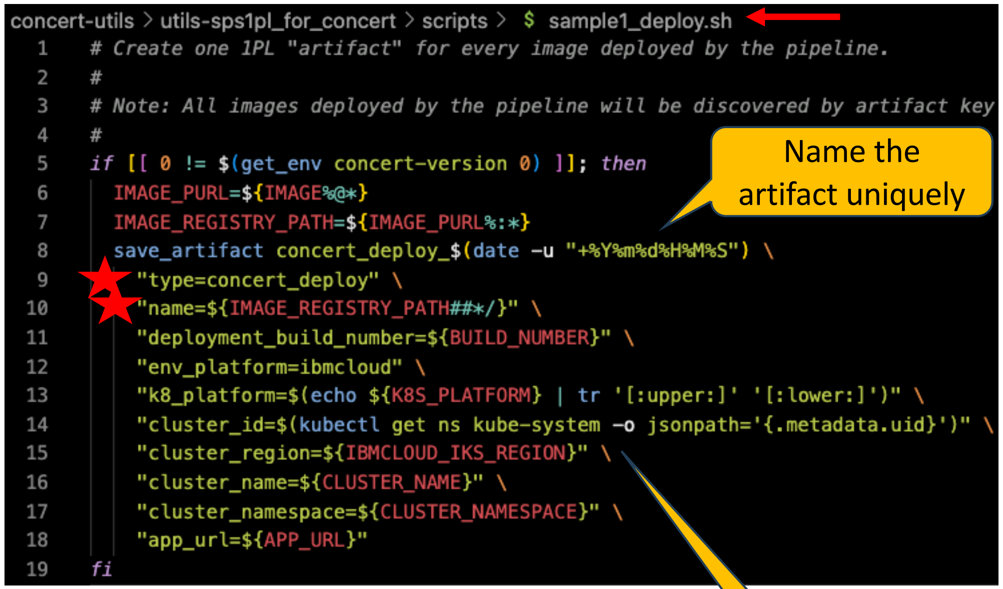
(Add to the end of common SPS/1PL file [./scripts/deploy.sh])

Note: concert_deploy artifacts should be excluded from SPS/1PL “collect-evidence” and “inventory add” tasks in [./scripts/run_test.sh] & [./scripts/release.sh], respectively).





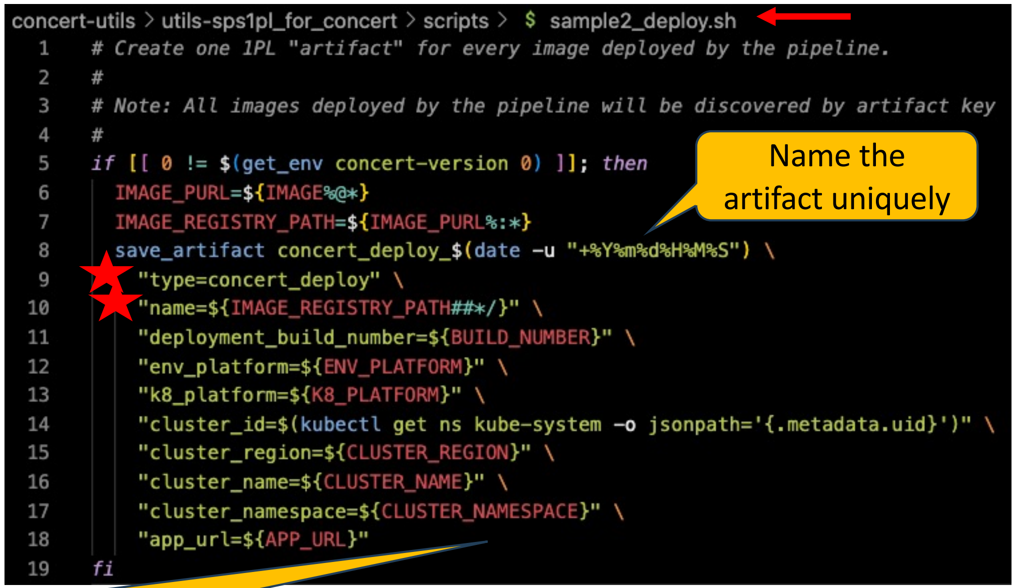
```
artifact_type="$(load_artifact "${artifact}" type)"
if [[ ${artifact_type} != "concert_deploy" ]]; then
  params+=(--assets "$artifact":"artifact")
fi
```



concert-utils > utils-sps1pl_for_concert > scripts > $ sample1_deploy.sh

```
# Create one 1PL "artifact" for every image deployed by the pipeline.
#
# Note: All images deployed by the pipeline will be discovered by artifact key
#
if [[ 0 != $(get_env concert-version 0) ]]; then
  IMAGE_PURL=${IMAGE%@*}
  IMAGE_REGISTRY_PATH=${IMAGE_PURL%:*}
  save_artifact concert_deploy_$(date -u "+%Y%m%d%H%M%S") \
    "type=concert_deploy" \
    "name=${IMAGE_REGISTRY_PATH##*/}" \
    "deployment_build_number=${BUILD_NUMBER}" \
    "env_platform=ibmcloud" \
    "k8_platform=$(echo ${K8S_PLATFORM} | tr '[:upper:]' '[:lower:]')" \
    "cluster_id=$(kubectl get ns kube-system -o jsonpath='{.metadata.uid}')" \
    "cluster_region=${IBMCLOUD_IKS_REGION}" \
    "cluster_name=${CLUSTER_NAME}" \
    "cluster_namespace=${CLUSTER_NAMESPACE}" \
    "app_url=${APP_URL}"
fi
```

Name the artifact uniquely



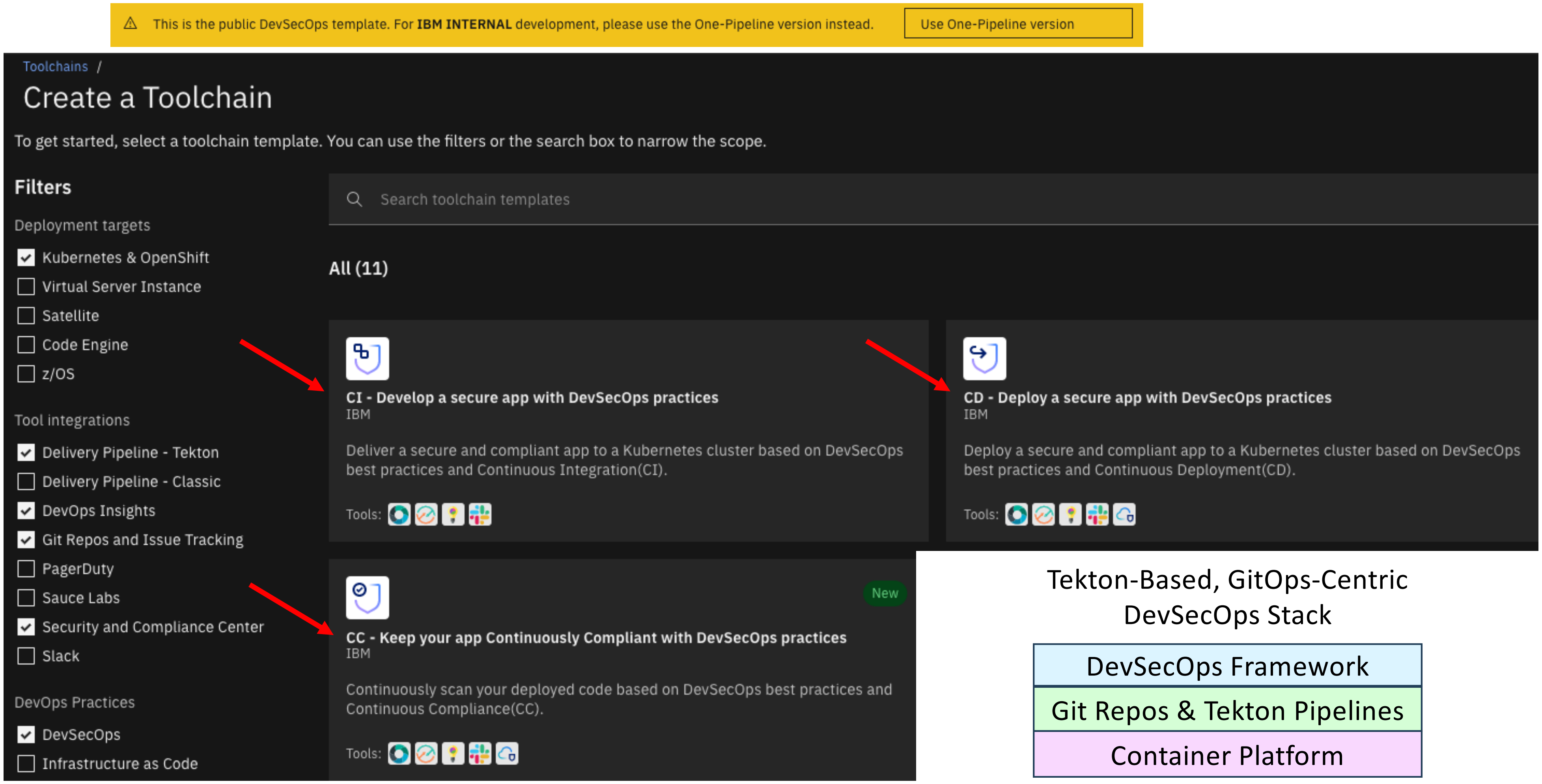
concert-utils > utils-sps1pl_for_concert > scripts > $ sample2_deploy.sh

```
# Create one 1PL "artifact" for every image deployed by the pipeline.
#
# Note: All images deployed by the pipeline will be discovered by artifact key
#
if [[ 0 != $(get_env concert-version 0) ]]; then
  IMAGE_PURL=${IMAGE%@*}
  IMAGE_REGISTRY_PATH=${IMAGE_PURL%:*}
  save_artifact concert_deploy_$(date -u "+%Y%m%d%H%M%S") \
    "type=concert_deploy" \
    "name=${IMAGE_REGISTRY_PATH##*/}" \
    "deployment_build_number=${BUILD_NUMBER}" \
    "env_platform=${ENV_PLATFORM}" \
    "k8_platform=${K8_PLATFORM}" \
    "cluster_id=$(kubectl get ns kube-system -o jsonpath='{.metadata.uid}')" \
    "cluster_region=${CLUSTER_REGION}" \
    "cluster_name=${CLUSTER_NAME}" \
    "cluster_namespace=${CLUSTER_NAMESPACE}" \
    "app_url=${APP_URL}"
fi
```

Name the artifact uniquely

Different sets of environment variables were used for deploying images.

DevSecOps Toolchains on IBM Cloud: CI, CD, CC (Instantiated via Wizards)



⚠ This is the public DevSecOps template. For **IBM INTERNAL** development, please use the One-Pipeline version instead. | Use One-Pipeline version

Toolchains /

Create a Toolchain

To get started, select a toolchain template. You can use the filters or the search box to narrow the scope.

Filters

Deployment targets

- [✓ Kubernetes & OpenShift]
- Virtual Server Instance
- Satellite
- Code Engine
- z/OS

Tool integrations

- [✓ Delivery Pipeline - Tekton]
- Delivery Pipeline - Classic
- [✓ DevOps Insights]
- [✓ Git Repos and Issue Tracking]
- PagerDuty
- Sauce Labs
- [✓ Security and Compliance Center]
- Slack

DevOps Practices

- [✓ DevSecOps]
- Infrastructure as Code

Search toolchain templates

All (11)

CI - Develop a secure app with DevSecOps practices
IBM
Deliver a secure and compliant app to a Kubernetes cluster based on DevSecOps best practices and Continuous Integration(CI).
Tools:

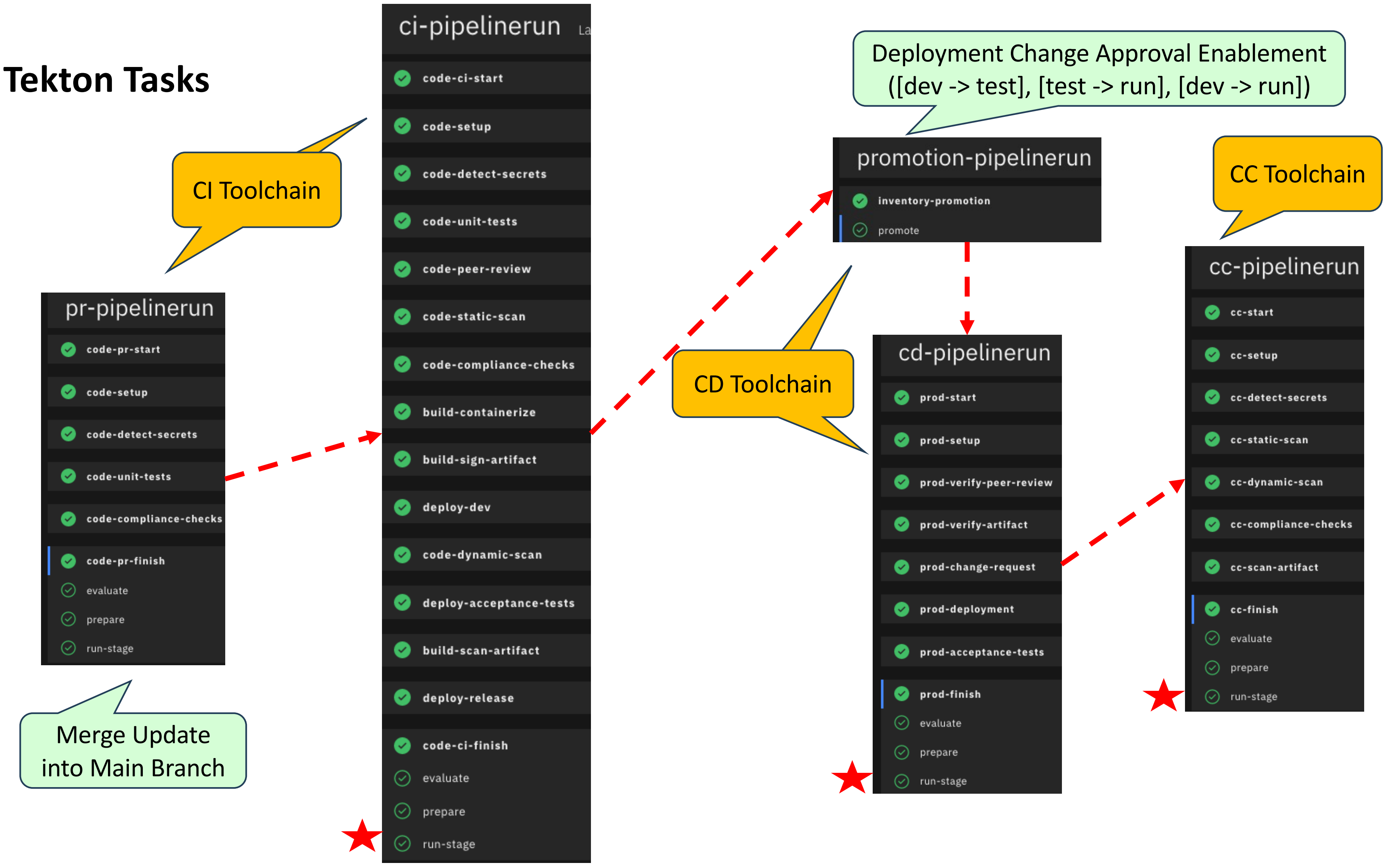
CD - Deploy a secure app with DevSecOps practices
IBM
Deploy a secure and compliant app to a Kubernetes cluster based on DevSecOps best practices and Continuous Deployment(CD).
Tools:

New
CC - Keep your app Continuously Compliant with DevSecOps practices
IBM
Continuously scan your deployed code based on DevSecOps best practices and Continuous Compliance(CC).
Tools:

Tekton-Based, GitOps-Centric
DevSecOps Stack

DevSecOps Framework
Git Repos & Tekton Pipelines
Container Platform

Tekton Tasks


CI Toolchain
Merge Update into Main Branch
Deployment Change Approval Enablement ([dev -> test], [test -> run], [dev -> run])
CD Toolchain
CC Toolchain

pr-pipelinerun
- code-pr-start
- code-setup
- code-detect-secrets
- code-unit-tests
- code-compliance-checks
- code-pr-finish
- evaluate
- prepare
- run-stage

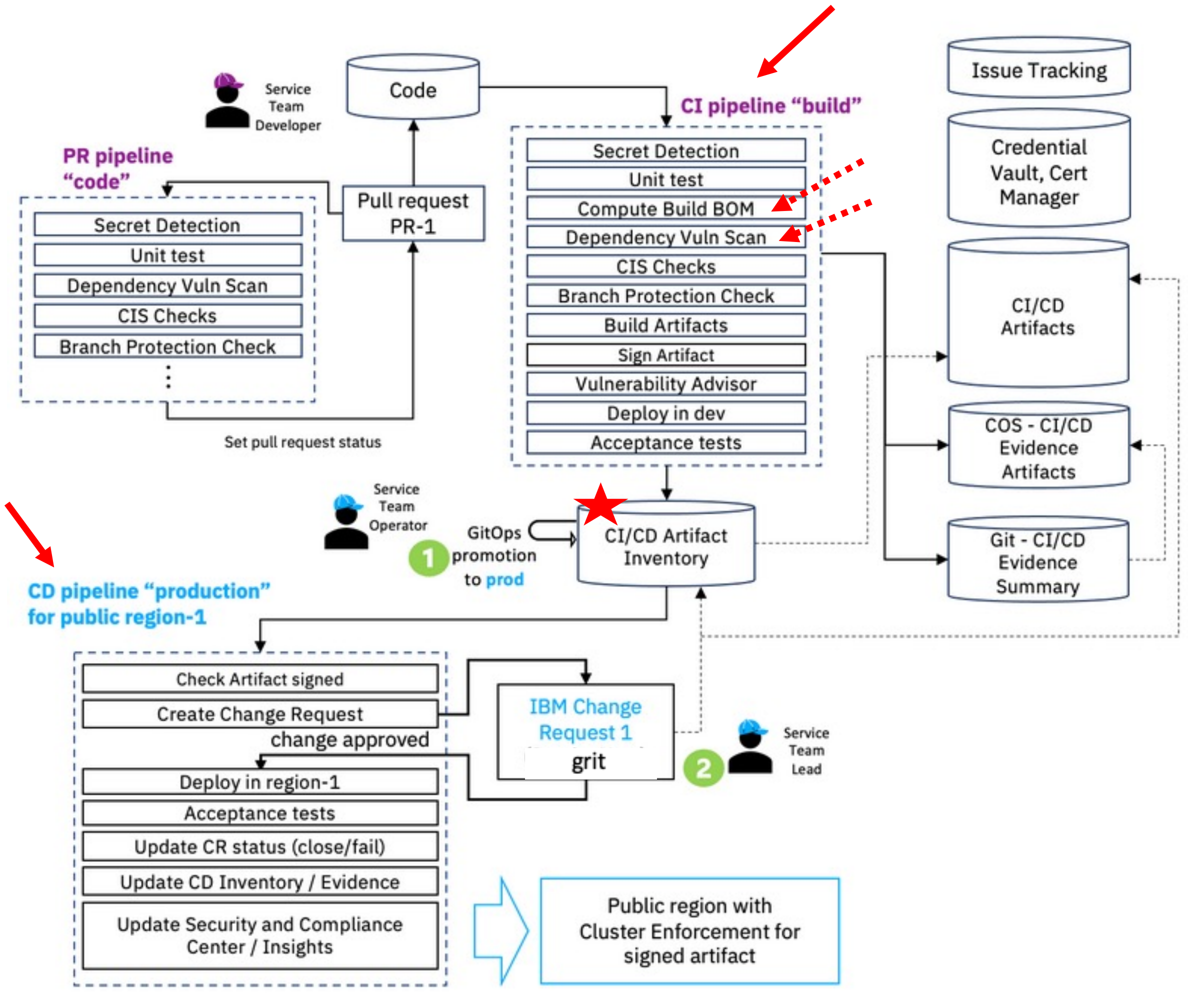
ci-pipelinerun
- code-ci-start
- code-setup
- code-detect-secrets
- code-unit-tests
- code-peer-review
- code-static-scan
- code-compliance-checks
- build-containerize
- build-sign-artifact
- deploy-dev
- code-dynamic-scan
- deploy-acceptance-tests
- build-scan-artifact
- deploy-release
- code-ci-finish
- evaluate
- prepare
- run-stage

promotion-pipelinerun
- inventory-promotion
- promote

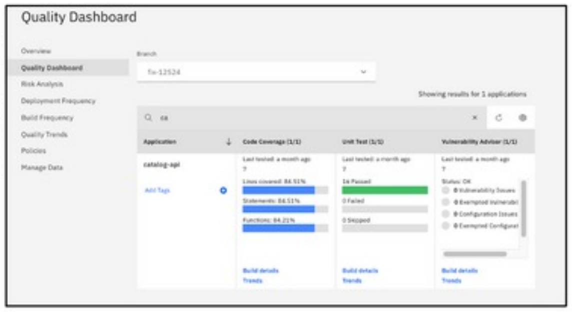
cd-pipelinerun
- prod-start
- prod-setup
- prod-verify-peer-review
- prod-verify-artifact
- prod-change-request
- prod-deployment
- prod-acceptance-tests
- prod-finish
- evaluate
- prepare
- run-stage

cc-pipelinerun
- cc-start
- cc-setup
- cc-detect-secrets
- cc-static-scan
- cc-dynamic-scan
- cc-compliance-checks
- cc-scan-artifact
- cc-finish
- evaluate
- prepare
- run-stage

DevSecOps Architecture: https://test.cloud.ibm.com/docs/devsecops?topic=devsecops-cd-devsecops-arch



DevOps Insights



Git Issues

Change Request

- Cumulative list of changes
- Traceability: Issue/Pull Request
- Bill of material
- Status on each control:
  - ✓ Secret detection
  - ✓ Unit test passed
  - ✓ Dependency code vulnerability scan (CVE)
  - ✓ Signed artifact
  - ✓ Binary vulnerability scan – VA (CVE)
  - ✓ Acceptance tests in PRE-PROD
  - ✓ …
- Links to Evidence (COS)

Security and Compliance Center