To convert a JSON file to YAML in an automation workflow and push the YAML file to GitHub, you can use a script or a CI/CD pipeline. Here's a full breakdown of how to do it using a **Python script** and **Git commands**, which is common in infrastructure automation:

**✅ Step-by-Step: Convert JSON to YAML and Push to GitHub**

**1. Install Required Tools**

Make sure the system has:

* python3
* PyYAML (for conversion)
* git (for pushing to GitHub)

Install PyYAML:

bash

CopyEdit

pip install pyyaml

**2. Python Script: Convert JSON to YAML**

Save this script as convert\_json\_to\_yaml.py:

python

CopyEdit

import json

import yaml

import sys

# Load JSON file

with open('input.json', 'r') as json\_file:

data = json.load(json\_file)

# Save as YAML

with open('output.yaml', 'w') as yaml\_file:

yaml.dump(data, yaml\_file, default\_flow\_style=False)

print("Conversion complete: output.yaml")

**3. Automate via Shell Script (Optional)**

bash

CopyEdit

#!/bin/bash

# Step 1: Run Python conversion

python3 convert\_json\_to\_yaml.py

# Step 2: Git operations

git add output.yaml

git commit -m "Add converted YAML from JSON"

git push origin main

✅ **Make executable**:

bash

CopyEdit

chmod +x convert\_and\_push.sh

**4. If Using Tekton or GitHub Actions CI/CD**

You can add this step in a **Tekton Task** or **GitHub Actions** job with:

* A step to run Python
* A step to push using a token-based Git push

Let me know if you want that script in **Tekton YAML**, **GitHub Actions YAML**, or **Ansible**.

**🧠 Bonus Tip:**

If you automate many such conversions:

* Use .gitignore to avoid accidental pushes of temp files.
* Validate YAML syntax using tools like yamllint.

Would you like this flow converted into a **Tekton Task** or GitHub Actions workflow as well?

**You said:**

tekton task

**ChatGPT said:**

Here’s how to create a **Tekton Task** to convert a JSON file to YAML and push it to GitHub. This includes:

* A Tekton **Task** definition
* A **PipelineRun** to invoke it
* A **ServiceAccount** with GitHub push access

**✅ Tekton Task: Convert JSON to YAML & Push to GitHub**

**1. Create the Task (json-to-yaml-task.yaml)**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: Task

metadata:

name: json-to-yaml-task

spec:

params:

- name: repo-url

type: string

- name: git-username

type: string

- name: git-email

type: string

workspaces:

- name: shared-data

steps:

- name: install-packages

image: python:3.11

script: |

pip install pyyaml

- name: convert-json-to-yaml

image: python:3.11

workingDir: $(workspaces.shared-data.path)

script: |

pip install pyyaml

python -c '

import json, yaml

with open("input.json") as f:

data = json.load(f)

with open("output.yaml", "w") as f:

yaml.dump(data, f, default\_flow\_style=False)

print("Converted input.json to output.yaml")

'

- name: git-commit-push

image: alpine/git

workingDir: $(workspaces.shared-data.path)

script: |

git config --global user.name "$(params.git-username)"

git config --global user.email "$(params.git-email)"

git init

git remote add origin $(params.repo-url)

git add output.yaml

git commit -m "Add YAML converted from JSON via Tekton"

git push origin HEAD:main

**2. Create the PipelineRun (json-to-yaml-pipelinerun.yaml)**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: PipelineRun

metadata:

name: json-to-yaml-run

spec:

taskRef:

name: json-to-yaml-task

workspaces:

- name: shared-data

persistentVolumeClaim:

claimName: my-shared-pvc # Create or use existing PVC

params:

- name: repo-url

value: https://<your-token>@github.com/youruser/your-repo.git

- name: git-username

value: your-git-username

- name: git-email

value: your@email.com

🔐 Replace <your-token> in repo-url with a GitHub Personal Access Token (PAT) for authentication.

**3. Ensure PVC Exists (my-shared-pvc.yaml)**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-shared-pvc

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

**🛡 Optional: Use a Git Secret with a ServiceAccount**

Let me know if you want to securely mount the Git credentials using a Kubernetes Secret and ssh-agent/HTTPS.

Would you like me to generate a **full Tekton Pipeline** with multiple tasks next (e.g. validation, ArgoCD sync, etc.)?

**You said:**

yes secret with a service account and full tekton pipeline

**ChatGPT said:**

Great! Here's a **complete Tekton CI pipeline** that:

1. Uses a **Kubernetes Secret** to securely store GitHub credentials
2. Uses a **ServiceAccount** that mounts the Secret
3. Includes:
   * JSON → YAML conversion
   * Git commit + push
4. Can be extended to trigger ArgoCD sync or other follow-up tasks

**✅ Overview of Components**

| **Resource** | **Purpose** |
| --- | --- |
| Secret | Stores GitHub credentials (username & PAT) |
| ServiceAccount | Used by Tekton tasks to access GitHub securely |
| Task | Converts JSON to YAML & pushes to GitHub |
| Pipeline | Chains the task(s) together |
| PipelineRun | Executes the pipeline |
| PVC | Shared workspace |

**1. 🔐 Secret for GitHub Credentials**

yaml

CopyEdit

apiVersion: v1

kind: Secret

metadata:

name: github-secret

annotations:

tekton.dev/git-0: https://github.com

type: kubernetes.io/basic-auth

stringData:

username: your-github-username

password: ghp\_xxxxxxYOURTOKENxxxxx

**2. 👤 ServiceAccount with Git Secret**

yaml

CopyEdit

apiVersion: v1

kind: ServiceAccount

metadata:

name: github-sa

secrets:

- name: github-secret

**3. 📦 PVC for Shared Workspace**

yaml

CopyEdit

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: shared-pvc

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 500Mi

**4. 🧪 Tekton Task: Convert JSON to YAML & Push**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: Task

metadata:

name: json-to-yaml-and-push

spec:

workspaces:

- name: shared-workspace

steps:

- name: convert-json-to-yaml

image: python:3.11

workingDir: $(workspaces.shared-workspace.path)

script: |

pip install pyyaml

python -c '

import json, yaml

with open("input.json") as f:

data = json.load(f)

with open("output.yaml", "w") as f:

yaml.dump(data, f, default\_flow\_style=False)

print("✅ JSON converted to YAML.")

'

- name: git-push

image: alpine/git

workingDir: $(workspaces.shared-workspace.path)

env:

- name: GIT\_USERNAME

valueFrom:

secretKeyRef:

name: github-secret

key: username

- name: GIT\_PASSWORD

valueFrom:

secretKeyRef:

name: github-secret

key: password

script: |

git config --global user.email "bot@example.com"

git config --global user.name "$GIT\_USERNAME"

git init

git remote add origin https://$GIT\_USERNAME:$GIT\_PASSWORD@github.com/<your-user>/<your-repo>.git

git add output.yaml

git commit -m "Tekton: Converted JSON to YAML"

git push origin HEAD:main

**5. 🔀 Pipeline**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: Pipeline

metadata:

name: json-to-yaml-pipeline

spec:

workspaces:

- name: shared-workspace

tasks:

- name: convert-and-push

taskRef:

name: json-to-yaml-and-push

workspaces:

- name: shared-workspace

workspace: shared-workspace

**6. 🚀 PipelineRun**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: PipelineRun

metadata:

name: json-to-yaml-run

spec:

pipelineRef:

name: json-to-yaml-pipeline

workspaces:

- name: shared-workspace

persistentVolumeClaim:

claimName: shared-pvc

serviceAccountName: github-sa

**🔄 Optional Add-on: ArgoCD Sync as Next Task**

Let me know if you'd like to:

* Trigger ArgoCD sync via CLI or webhook in a follow-up task
* Chain GitOps pipelines (multi-env deploys)

Would you like me to bundle all these as downloadable YAMLs or a Git repo structure?

**You said:**

argocd sync

**ChatGPT said:**

Perfect. To extend your **Tekton Pipeline** with an **ArgoCD sync step**, we’ll add a new task *after pushing the YAML to GitHub*.

This **ArgoCD sync task** will trigger an application sync using the ArgoCD CLI (argocd app sync <app-name>), usually run inside a container that has the CLI installed and access to the ArgoCD server.

**✅ Prerequisites for ArgoCD Sync**

1. **Access Token for ArgoCD** (or use argocd login via credentials)
2. The argocd CLI must be available in the image used by the step
3. Network access to the ArgoCD API server from within the Tekton pod

**🔐 Secret for ArgoCD Auth (optional)**

yaml

CopyEdit

apiVersion: v1

kind: Secret

metadata:

name: argocd-secret

type: Opaque

stringData:

ARGOCD\_AUTH\_TOKEN: <your-argocd-token>

**🧩 Add ArgoCD Sync Task (Tekton Task)**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: Task

metadata:

name: argocd-sync-task

spec:

params:

- name: app-name

type: string

- name: argocd-server

type: string

steps:

- name: argocd-sync

image: argoproj/argocd:latest

env:

- name: ARGOCD\_AUTH\_TOKEN

valueFrom:

secretKeyRef:

name: argocd-secret

key: ARGOCD\_AUTH\_TOKEN

script: |

argocd login $(params.argocd-server) --insecure --auth-token $ARGOCD\_AUTH\_TOKEN

argocd app sync $(params.app-name)

echo "✅ ArgoCD sync triggered for app: $(params.app-name)"

**🔀 Update Pipeline to Include ArgoCD Sync**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: Pipeline

metadata:

name: json-to-yaml-pipeline

spec:

workspaces:

- name: shared-workspace

params:

- name: app-name

type: string

- name: argocd-server

type: string

tasks:

- name: convert-and-push

taskRef:

name: json-to-yaml-and-push

workspaces:

- name: shared-workspace

workspace: shared-workspace

- name: sync-with-argocd

taskRef:

name: argocd-sync-task

runAfter:

- convert-and-push

params:

- name: app-name

value: $(params.app-name)

- name: argocd-server

value: $(params.argocd-server)

**🚀 PipelineRun Example with ArgoCD Sync**

yaml

CopyEdit

apiVersion: tekton.dev/v1

kind: PipelineRun

metadata:

name: json-to-yaml-run

spec:

pipelineRef:

name: json-to-yaml-pipeline

workspaces:

- name: shared-workspace

persistentVolumeClaim:

claimName: shared-pvc

serviceAccountName: github-sa

params:

- name: app-name

value: my-argocd-app

- name: argocd-server

value: argocd.mycompany.com

**✅ What This Does:**

* Converts input.json to output.yaml
* Commits and pushes to GitHub
* **Triggers argocd app sync** for my-argocd-app using a token

Would you like me to give all of these as a downloadable Git repo folder structure with Makefile/scripts?

**You said:**

yes

**ChatGPT said:**

✅ All Tekton pipeline files have been generated and packaged in the directory:  
**tekton-json-to-yaml-argocd**

You can download them here as a bundle:

🔗 Download Tekton JSON→YAML + ArgoCD Pipeline Folder

**📂 Contents**

| **File** | **Description** |
| --- | --- |
| 01-secret-github.yaml | GitHub credentials (username + PAT) |
| 02-secret-argocd.yaml | ArgoCD auth token secret |
| 03-serviceaccount.yaml | ServiceAccount using both secrets |
| 04-pvc.yaml | Shared PVC for Tekton workspace |
| 05-task-json-to-yaml.yaml | Task: Convert JSON → YAML and push to Git |
| 06-task-argocd-sync.yaml | Task: Trigger ArgoCD sync |
| 07-pipeline.yaml | Pipeline chaining both tasks |
| 08-pipelinerun.yaml | PipelineRun to execute the whole flow |

Let me know if you'd like to:

* Add validation/linting before push
* Trigger from a Git webhook or ArgoCD App-of-Apps
* Deploy this as part of GitOps with ArgoCD itself

# Converting JSON to YAML and Storing in a Tekton Pipeline

Here's how to convert JSON to YAML and use it in a Tekton pipeline:

## Option 1: Using a Task to Convert and Store

YAML

apiVersion: tekton.dev/v1beta1

kind: Pipeline

metadata:

name: json-to-yaml-pipeline

spec:

tasks:

- name: convert-json-to-yaml

taskRef:

name: json-to-yaml-converter

params:

- name: json-input

value: |

{

"name": "example",

"value": 42,

"nested": {

"key": "value"

}

}

workspaces:

- name: output

workspace: yaml-output

workspaces:

- name: yaml-output

## Option 2: Direct YAML in Pipeline

If you already have the YAML version, you can include it directly:

YAML

apiVersion: tekton.dev/v1beta1

kind: Pipeline

metadata:

name: example-pipeline

spec:

params:

- name: yaml-config

type: string

default: |

name: example

value: 42

nested:

key: value

tasks:

- name: use-yaml

taskRef:

name: some-task

params:

- name: config

value: $(params.yaml-config)

## Conversion Task Example

Here's a Task that performs the conversion:

YAML

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: json-to-yaml-converter

spec:

params:

- name: json-input

type: string

description: "JSON input to convert to YAML"

workspaces:

- name: output

description: "Where to write the YAML output"

steps:

- name: convert

image: python:3.9-alpine

script: |

#!/usr/bin/env python3

import json

import yaml

import sys

json\_data = """$(params.json-input)"""

data = json.loads(json\_data)

yaml\_output = yaml.dump(data, default\_flow\_style=False)

output\_path = "$(workspaces.output.path)/output.yaml"

with open(output\_path, 'w') as f:

f.write(yaml\_output)

## Using the Converted YAML

After conversion, you can use the YAML file in subsequent tasks:

YAML

- name: use-converted-yaml

taskRef:

name: yaml-processor

runAfter:

- convert-json-to-yaml

workspaces:

- name: input

workspace: yaml-output

Note: For the Python conversion, you'll need to ensure the pyyaml package is installed in your container image.



**Sarulnathan**6 minutes ago

how do you integrate python with tekton rehdat openshift

**DeepSeek-V3-0324**6 minutes ago

# Integrating Python with Tekton on Red Hat OpenShift

To integrate Python scripts with Tekton pipelines on OpenShift, you have several approaches. Here's a comprehensive guide:

## 1. Using Python in Tekton Tasks

### Basic Python Task Example

YAML

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: python-task

spec:

steps:

- name: run-python-script

image: python:3.9-slim # Official Python image or use UBI-based image for OpenShift

script: |

#!/usr/bin/env python3

print("Hello from Python in Tekton!")

# Your Python code here

import os

print(f"Current directory: {os.getcwd()}")

### OpenShift-optimized Python Task (using UBI)

YAML

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: python-ubi-task

spec:

steps:

- name: run-python

image: registry.access.redhat.com/ubi8/python-39 # Red Hat UBI Python image

script: |

#!/usr/bin/env python3

print("Running Python on OpenShift UBI")

# Install additional packages if needed

pip install requests pandas --user

# Your Python script

## 2. Passing Parameters and Workspaces

YAML

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: python-data-processor

spec:

params:

- name: input-data

type: string

workspaces:

- name: shared-data

steps:

- name: process-data

image: registry.access.redhat.com/ubi8/python-39

script: |

#!/usr/bin/env python3

import json

import yaml

import os

# Access Tekton parameters

input\_data = """$(params.input-data)"""

print(f"Received input: {input\_data}")

# Access workspace files

workspace\_path = "$(workspaces.shared-data.path)"

output\_file = os.path.join(workspace\_path, "result.json")

# Process data

data = {"processed": True, "input": input\_data}

with open(output\_file, 'w') as f:

json.dump(data, f)

## 3. Python in a Pipeline

YAML

apiVersion: tekton.dev/v1beta1

kind: Pipeline

metadata:

name: python-pipeline

spec:

params:

- name: python-script

type: string

workspaces:

- name: shared-workspace

tasks:

- name: run-python

taskRef:

name: python-task

params:

- name: script

value: $(params.python-script)

workspaces:

- name: shared-data

workspace: shared-workspace

## 4. Best Practices for OpenShift

1. **Use UBI-based images**:

YAML

image: registry.access.redhat.com/ubi8/python-39

1. **Manage dependencies**:

Python

# In your script:

import subprocess

subprocess.run(["pip", "install", "-r", "requirements.txt", "--user"], check=True)

1. **Handle OpenShift security**:
   * Ensure your service account has proper permissions
   * Run as non-root user when possible:

YAML

securityContext:

runAsNonRoot: true

runAsUser: 1001

## 5. Complete Pipeline Example

YAML

apiVersion: tekton.dev/v1beta1

kind: Pipeline

metadata:

name: python-data-pipeline

spec:

workspaces:

- name: shared-data

params:

- name: input-json

type: string

tasks:

- name: process-with-python

taskRef:

name: python-data-processor

params:

- name: input-data

value: $(params.input-json)

workspaces:

- name: shared-data

workspace: shared-workspace

- name: next-step

runAfter: ["process-with-python"]

taskRef:

name: another-task

workspaces:

- name: input

workspace: shared-workspace

## 6. Building Custom Python Images

For complex dependencies, build a custom image:

Dockerfile

FROM registry.access.redhat.com/ubi8/python-39

USER 0

COPY requirements.txt .

RUN pip install -r requirements.txt

USER 1001

COPY script.py /opt/script.py

ENTRYPOINT ["python", "/opt/script.py"]

Then reference it in your Task:

YAML

image: image-registry.openshift-image-registry.svc:5000/your-namespace/custom-python-image

## Troubleshooting Tips

1. Check Python version in the container:

Python

import sys

print(sys.version)

1. Verify installed packages:

Python

import pip

print(pip.get\_installed\_distributions())

1. Check file permissions in workspaces:

Python

import os

print(os.listdir('$(workspaces.shared-data.path)'))

This approach gives you a flexible way to integrate Python scripts with your Tekton pipelines on OpenShift while following Red Hat best practices.



**Sarulnathan**2 minutes ago

CI / CD application intergration from github to redhat openshift via tekton and argocd in RHEL

**DeepSeek-V3-0324**1 minute ago

# CI/CD Pipeline from GitHub to Red Hat OpenShift using Tekton and ArgoCD

Here's a comprehensive guide to set up a complete GitOps workflow from GitHub to OpenShift using Tekton for CI and ArgoCD for CD.

## Architecture Overview

Code

GitHub (Source Code)

→ Tekton (CI: Build & Test)

→ OpenShift Image Registry

→ ArgoCD (CD: Deployment)

→ OpenShift Cluster

## Prerequisites

1. Red Hat OpenShift 4.x cluster with admin access
2. OpenShift Pipelines (Tekton) operator installed
3. OpenShift GitOps (ArgoCD) operator installed
4. GitHub repository with application code
5. oc CLI and tkn CLI installed on your RHEL system

## 1. Set Up Tekton Pipeline (CI)

### Create Pipeline Resources

YAML

# pipeline-resources.yaml

apiVersion: tekton.dev/v1alpha1

kind: PipelineResource

metadata:

name: app-source-code

spec:

type: git

params:

- name: url

value: https://github.com/your-org/your-app-repo

- name: revision

value: main

---

apiVersion: tekton.dev/v1alpha1

kind: PipelineResource

metadata:

name: app-image

spec:

type: image

params:

- name: url

value: image-registry.openshift-image-registry.svc:5000/your-namespace/your-app-image

### Create Tekton Tasks

YAML

# build-task.yaml

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: build-app

spec:

params:

- name: context

description: Path to build context

default: .

workspaces:

- name: source

steps:

- name: build

image: registry.access.redhat.com/ubi8/buildah

workingDir: $(workspaces.source.path)

script: |

#!/usr/bin/env bash

buildah bud -f Dockerfile -t $(resources.outputs.image.url) $(params.context)

buildah push $(resources.outputs.image.url)

securityContext:

privileged: true

### Create Tekton Pipeline

YAML

# app-pipeline.yaml

apiVersion: tekton.dev/v1beta1

kind: Pipeline

metadata:

name: app-ci-pipeline

spec:

resources:

- name: source-code

type: git

- name: image

type: image

workspaces:

- name: shared-workspace

tasks:

- name: fetch-source

taskRef:

name: git-clone

workspaces:

- name: output

workspace: shared-workspace

params:

- name: url

value: $(resources.source-code.url)

- name: revision

value: $(resources.source-code.revision)

- name: build-image

taskRef:

name: build-app

runAfter: ["fetch-source"]

workspaces:

- name: source

workspace: shared-workspace

resources:

outputs:

- name: image

resource: image

## 2. Set Up ArgoCD (CD)

### Create ArgoCD Application

YAML

# argocd-app.yaml

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: your-app

namespace: openshift-gitops

spec:

destination:

server: https://kubernetes.default.svc

namespace: your-target-namespace

project: default

source:

path: k8s/ # Path to your Kubernetes manifests in the repo

repoURL: https://github.com/your-org/your-app-repo

targetRevision: HEAD

helm:

parameters:

- name: image.tag

value: latest # This will be updated by the pipeline

syncPolicy:

automated:

prune: true

selfHeal: true

syncOptions:

- CreateNamespace=true

## 3. Set Up GitHub Webhook (Optional)

bash

# Get webhook URL from Tekton EventListener

WEBHOOK\_URL=$(oc get route el-github-listener -n your-tekton-namespace -o jsonpath='{.spec.host}')

# Configure GitHub webhook

curl -X POST \

-H "Authorization: token YOUR\_GITHUB\_TOKEN" \

-H "Accept: application/vnd.github.v3+json" \

https://api.github.com/repos/your-org/your-repo/hooks \

-d '{

"name": "web",

"active": true,

"events": ["push"],

"config": {

"url": "http://'$WEBHOOK\_URL'",

"content\_type": "json"

}

}'

## 4. Complete CI/CD Workflow

1. **Developer pushes code to GitHub**
2. **Tekton Pipeline triggers**:
   * Clones the repository
   * Builds container image
   * Pushes to OpenShift internal registry
   * Updates the image tag in Git (optional)
3. **ArgoCD detects changes**:
   * Pulls updated manifests from Git
   * Deploys new version to OpenShift
   * Health checks and rollback if needed

## 5. Advanced Configuration

### Image Tag Update Task

YAML

# update-image-tag-task.yaml

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: update-image-tag

spec:

params:

- name: new-tag

type: string

- name: git-repo

type: string

- name: git-branch

type: string

default: "main"

workspaces:

- name: source

- name: git-credentials

steps:

- name: update-tag

image: alpine/git

workingDir: $(workspaces.source.path)

script: |

#!/bin/sh

git config --global user.email "tekton@example.com"

git config --global user.name "Tekton Bot"

# Update image tag in deployment.yaml

sed -i "s|image:.\*|image: $(params.new-tag)|" k8s/deployment.yaml

git add k8s/deployment.yaml

git commit -m "Update image tag to $(params.new-tag)"

git push origin $(params.git-branch)

### Tekton TriggerTemplate

YAML

# trigger-template.yaml

apiVersion: triggers.tekton.dev/v1beta1

kind: TriggerTemplate

metadata:

name: github-push-template

spec:

params:

- name: git-revision