

Python Version Compatibility Issues



```
#14 [10/11] RUN poetry install --only main
#14 2.527 The currently activated Python version 3.9.20 is not supported by the project (3.9.19).
#14 2.527 Trying to find and use a compatible version.
#14 3.235
#14 3.235 Poetry was unable to find a compatible version. If you have one, you can explicitly use it
via the "env use" command.
#14 ERROR: process "/bin/sh -c poetry install --only main" did not complete successfully: exit code:
1
```



The error you're encountering suggests that the version of Python (3.9.20) currently active in your environment is not supported by the project, which is explicitly looking for version 3.9.19. To resolve this, here are a few approaches you can try:

1. Specify the Correct Python Version

You can manually tell `poetry` to use the correct Python version (3.9.19) by adding this step to your Dockerfile or build process:

```
bash
```

```
RUN poetry env use 3.9.19
```

This command ensures `poetry` activates the correct Python environment version during the build process.

2. Adjust the Dockerfile to Install the Correct Python Version

If the Dockerfile does not have the correct Python version installed, update it to ensure Python 3.9.19 is installed. Here's an example of what you might need to add before the `poetry install` step:

```
bash
```

```
RUN apt-get update && apt-get install -y python3.9=3.9.19*
```

This ensures the correct version of Python is available.

3. Update the `pyproject.toml` to Accept Python 3.9.20

If you are okay with using Python 3.9.20, you can modify your `pyproject.toml` file to allow this version of Python by changing the `python` version specification:

```
toml
```

```
[tool.poetry.dependencies]
python = "^3.9" # This will allow 3.9.x versions including 3.9.20
```

This will allow `poetry` to use Python 3.9.20.

4. Manually Downgrade Python in the Docker Container

If the issue is related to a global Python version that is incompatible, you can explicitly install and use Python 3.9.19 before the `poetry install` command:

```
bash
```

```
RUN apt-get install -y python3.9=3.9.19*
```

Then, rerun the build process to ensure the correct Python version is used.
Let me know how it goes!



So this just started happening - I am using cdk and this is the first time I have run into this issue



Memory updated

It sounds like something may have changed in your environment or in the Docker image you are using. Since you're encountering this for the first time with CDK, it's possible that a recent update or a new Docker base image introduced the Python version mismatch. Here are a few steps you could try to diagnose and fix the issue:

1. Check Your Docker Base Image

- If you recently updated your Dockerfile or are using a different base image, verify that the base image supports the Python version you need (3.9.19).
- Check the Docker image version you're using and see if it was recently updated to include Python 3.9.20. If so, you may want to lock it to a specific version that still uses Python 3.9.19.

Example:

```
Dockerfile
```

```
FROM python:3.9.19-slim # Lock to a specific Python version
```

2. Look for CDK Version Updates

- If you've updated your AWS CDK dependencies, check if there are any changes that could be affecting how your build is configured, especially around the Docker images or Python environment.

3. Try Downgrading Poetry or Pinning Versions

- It's possible that a recent update to `poetry` or a new behavior in how it manages environments is causing this issue. You might want to explicitly install a stable version of `poetry` in your Dockerfile.

Example:

Dockerfile

```
RUN pip install poetry==1.4.0 # Replace with a stable version
```

4. Inspect Your CDK Stack Configuration

- Review your `cdk.json` or the CDK constructs to see if there have been any recent changes related to the Lambda or Fargate configurations that might have introduced new dependencies or environment variables affecting Python.

If none of these resolve the issue, there might have been some dependency changes within CDK or your project's dependencies. Let me know if any of these steps help!



I did delete cdk.out before running the last build



Memory updated

Deleting `cdk.out` should normally regenerate your stack outputs without causing issues, but it's possible that it led to some reconfiguration or reinstallation of dependencies, which could have triggered the Python version mismatch.

Here are a few things you can try:

1. Clean and Rebuild CDK

After deleting `cdk.out`, make sure to clean up other potential temporary files and force a full rebuild:

bash

```
cdk destroy
cdk synth
cdk deploy
```

This will ensure everything is fully regenerated from scratch.

2. Reinstall Dependencies

If the issue persists, reinstall all project dependencies:

```
bash

rm -rf .venv
poetry install
```

This ensures that you're working with a fresh environment.

3. Check for Conflicting Dependencies

Since deleting `cdk.out` regenerates stack outputs, check for any dependencies or configurations that may have been altered, especially if you were caching dependencies or using older versions.

Let me know if these steps help!



what does this do? I already have a python venv defined and it is used by other project components



Deleting `.venv` and reinstalling dependencies will only affect the virtual environment in the specific directory or project you're working in. Here's a breakdown of what the steps I mentioned do and their impact:

1. Deleting `.venv`

The `.venv` directory contains the virtual environment for your project. This directory includes a local installation of Python and all the packages specific to this project. Deleting it and reinstalling dependencies will:

- Remove the local Python environment specific to the current project.
- Clear all project-specific dependencies (like CDK, `poetry`, etc.).
- Force `poetry` or `pip` to reinstall all dependencies based on your `pyproject.toml` or `requirements.txt`.

Impact: It only affects the current project and won't interfere with other projects that use their own virtual environments.

2. Reinstalling Dependencies

After deleting `.venv`, running `poetry install` will:

- Recreate the `.venv` directory.
- Reinstall the exact versions of dependencies specified in your `pyproject.toml`.
- Ensure you're starting with a fresh environment, eliminating potential issues from conflicting package versions or an outdated cache.

Impact: This will not affect other project components that use a separate virtual environment. It will only reset dependencies for the specific project where you're running the command.

Why Is This Helpful?

- If your Python environment got into a state where dependency versions conflict or there is a mismatch (like the Python version issue you're seeing), starting fresh can resolve these issues.
- Reinstalling ensures all dependencies are compatible with the Python version defined in your project (3.9.19 in your case).

Since you already have a `.venv` defined, if you prefer not to delete .venv` , you can try running the following inside your existing environment to reinstall dependencies:`

```
bash
```

```
poetry install --no-root
```

This reinstalls dependencies without removing the environment.

Let me know if you'd like to explore another approach!

ChatGPT can make mistakes. Check important info.