

Project 2 final deliverable -- Report

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

Type or sign your names: Katie Roberts, Ashwin
Senthilkumar, Parker Bushey

Write today's date: 12/8/2021

Assignment

Fill out each of the following sections.

Location of project

Provide a URL that we can use to interact with your team's deployed service:

<https://api-dot-ece461-p2-t3.uc.r.appspot.com>

Provide a link to your team's code repository on GitHub:

<https://github.com/Purdue-ECE-461/project-2-3/api>

Succinct description

In a 5-7 sentence paragraph, describe the system that you have implemented.

Our restful API uses gcloud SDK and Cloud App Engine for deployment. The API endpoints are configured with Cloud Endpoints service. The code is written in Python. The API is made to store and manipulate data on a registry (Cloud Firestore collection) then perform computations on that data. This data (referred to as a package) represents an npm package or github repository by storing only key information such as the url, name, and version. Users of the API can further provide the content of the package in a b64 encoded string, or a JSProgram they want to associate with the package. The API will rate the trustworthiness of each package it is sent, and will not store the package if deemed untrustworthy.

Functional requirements

Baseline metric

You were required to implement a new metric to compute the degree of version pinning in a package. This implementation needed to build on another team's Project 1 implementation.

Describe any changes you made to the existing Project 1 design or implementation,¹ divided into two kinds:

1. Changes to allow you to implement the new metric

Change 1: New file SemverRange created

Justification: This file gets the dependency versions from a package.json then parses them to count the ratio of "pinned" dependencies / total number of dependencies. This ratio is the new metric goodPinningPractice. The new net score is then calculated as $0.9 * \text{old net score} + 0.1 * \text{goodPinningPractice}$.

¹ Hint: It might be helpful to examine your team's PRs or git logs to recall these changes. I assume, of course, that you followed an appropriate engineering process so that you can answer these questions.

- Changes to improve the reliability of the component so that your Project 2 implementation would satisfy the customer's requirements.

Change 1: File writes are directed to tmp directory

Justification: This was necessary to make the GCP deployment function, as a deployed instance is only allowed to write to it's own tmp directory. For more information, see <https://cloud.google.com/appengine/docs/standard/go111/using-temp-files>

Baseline API

In your *Project 2 Plan* document, you described the system features and requirements you planned to implement. Here you will describe how things went.

Fill in the following table for each of the *baseline behavioral features* (e.g. “ingest a package”) and the degree to which you’ve met each of them. Make one copy of the table per feature.

Feature: create/ingest a package

Relevant endpoint(s): <https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/>

How completely is it implemented? *complete*

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|--|---------------------------------|---|
| POST Valid input | Manual | packageCreateSuccess.png packageIngestSuccess.png FirestoreCreate.png |
| POST Valid input, package already exists in registry | Manual | ExistsAlready.png |
| POST invalid input | Manual | CreateInvalid.png |

*Validation approaches: For example, (None ; Manual ; Automated unit tests ; Automated end-to-end tests)

**Test records: For example, link to the relevant tests in your repository; link to the most recent relevant run of your CI; write the date of the most recent manual test.

** please see test records at <https://github.com/Purdue-ECE-461/project-2-3/doc/images>

Feature: package with id (Retrieve)

Relevant endpoint(s): <https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/<id>>

How completely is it implemented? *complete*

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|----------------------------|
| GET Valid input | Manual | PackageRetrieveSuccess.png |
| GET invalid input | Manual | IDNotExist.png |

Feature: package with id (Update)

Relevant endpoint(s): <https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/<id>>

How completely is it implemented? *complete*

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|--------------------------|
| PUT Valid input | Manual | PackageUpdateSuccess.png |
| PUT invalid input | Manual | UpdateNotExist.png |

Feature: package with id (Delete)

Relevant endpoint(s): <https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/<id>>

How completely is it implemented? *complete*

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|--|---------------------------------|-----------------------|
| DELETE Valid input | Manual | IDDeleteSuccess.png |
| DELETE invalid input, package does not exist in registry | Manual | IDDelNotExist.png |

Feature: package with Name (Retrieve)

Relevant endpoint(s):

<https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/ByName/<Name>>

How completely is it implemented? *complete*

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|-------------------------|
| GET Valid input | Manual | NameRetrieveSuccess.png |
| GET invalid input | Manual | NameNotExist.png |

Feature: package with Name (Delete)**Relevant endpoint(s):**<https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/ByName/<Name>>**How completely is it implemented?** *complete***How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|--|---------------------------------|-----------------------|
| DELETE Valid input | Manual | NameDeleteSuccess.png |
| DELETE invalid input, package does not exist in registry | Manual | NameDelNotExist.png |

Feature: rate a package**Relevant endpoint(s):** <https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/<id>/rate/>**How completely is it implemented?** *complete***How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|------------------------|
| GET Valid input | Manual | PackageRateSuccess.png |
| GET invalid input, package does not exist in registry | Manual | RateNotExist.png |
| GET Valid input, bad rating received | Manual | RateBad.png |

Feature: Package List**Relevant endpoint(s):** <https://api-dot-ece461-p2-t3.uc.r.appspot.com/packages/>**How completely is it implemented?** *incomplete, offset parameter doesn't affect anything***How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|------------------------|
| GET Valid input | Manual | PackageListSuccess.png |
| GET partially or all invalid input | Manual | |
| GET all packages “*” | Manual | |

Feature: Reset**Relevant endpoint(s):** <https://api-dot-ece461-p2-t3.uc.r.appspot.com/reset/>**How completely is it implemented?** *complete***How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|--|
| DELETE Valid input | Manual | ResetSuccess.png FirestoreReset.png |

Feature: authenticate**Relevant endpoint(s):** <https://api-dot-ece461-p2-t3.uc.r.appspot.com/authenticate/>**How completely is it implemented?** *partial: authenticate itself returns 501, however every other API request does check that x-auth exists in the request, and returns 400 (malformed request) if not found.***How did you validate it?** *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

| <i>Endpoint Verb(s) Payload option(s)</i> | <i>Validation approach(es)*</i> | <i>Test records**</i> |
|---|---------------------------------|-----------------------------------|
| GET Valid input | Manual | Auth.png |
| x-auth does not exist in request | Manual | NoAuth.png NoAuthMalformed.png |

Non-baseline: API

If you implemented additional behaviors in your team's web API, (e.g. package groups; traceability features; security features) they should be reflected in your team's OpenAPI specification on GitHub.

Provide a link to your team's OpenAPI specification:

<https://github.com/Purdue-ECE-461/project-2-3/api/api.json>

<https://www.getpostman.com/collections/daa4f5cfbf886fa57a0b>

Fill out the following template for each additional feature:

Feature: package History

Relevant endpoint(s) and input/output type(s):

<https://api-dot-ece461-p2-t3.uc.r.appspot.com/package/byName/<Name>>

returns package history list as a json

Summary of design:

when a package is created, updated, or rated, the action (HistoryEntry) object is created and appended to history as a json. A HistoryEntry object holds the type of action taken as an enum, the datetime the action was recorded, and a User object that will always be None since users were not implemented in our project.

How completely is it implemented? complete

How did you validate it? (Fill out this table for the feature – This should include the relevant kinds of error cases you tested)

see package with name (retrieve) validation above

Non-baseline: Browser-based interface

If you implemented a browser-based web interface, it should be ADA-compliant (WCAG 2.1 at level AA).

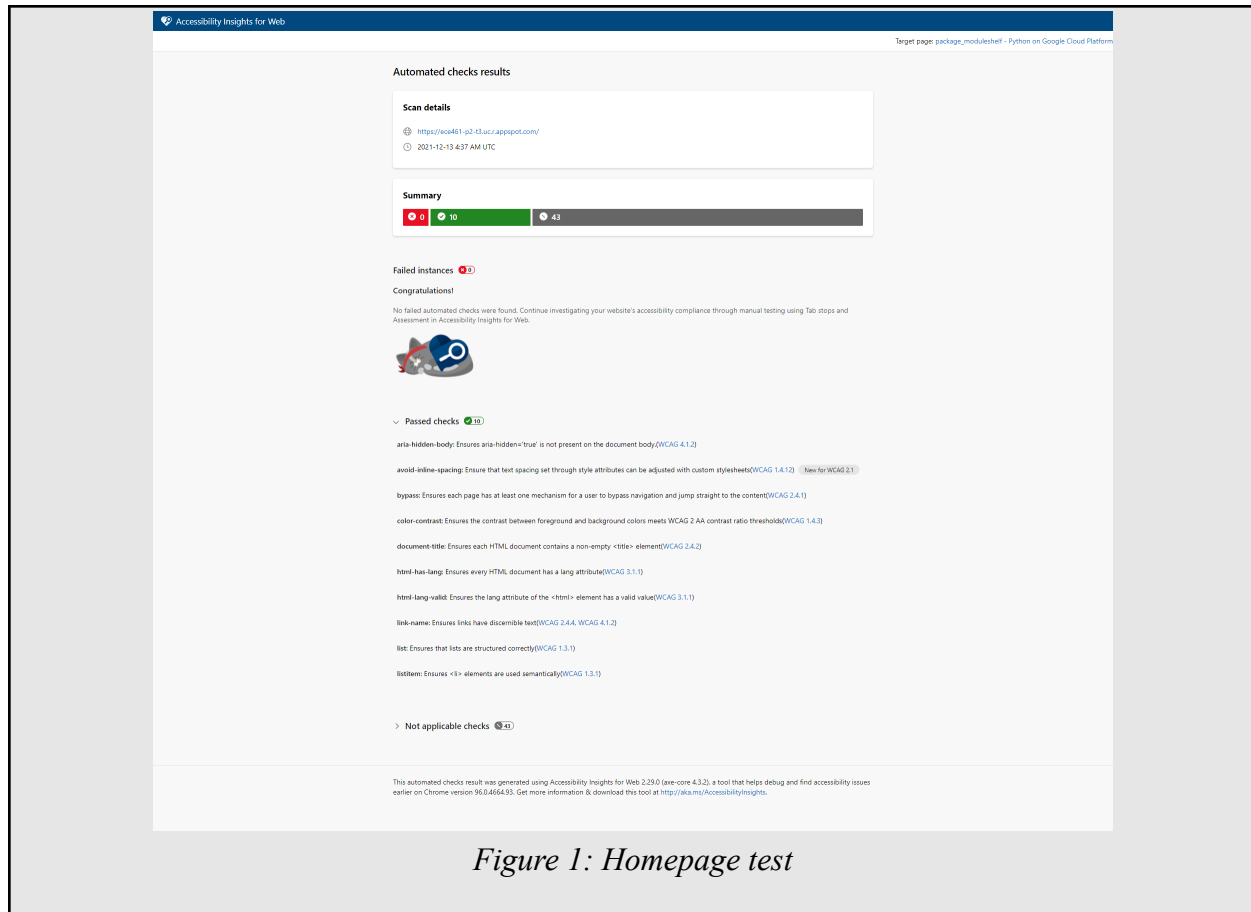
1. Show the result of the automated tests implemented by <https://github.com/microsoft/accessibility-insights-web>, as applied to each of your web page(s). Screenshots are fine.

Page 1: <https://ece461-p2-t3.uc.r.appspot.com>

This website is not complete. It is a clone of

<https://github.com/GoogleCloudPlatform/getting-started-python> with small modifications to have it deployed with our cloud service. You can see entries in the registry (firestore collection) there, but they will not be formatted correctly for viewing in full, and trying to use the site to create packages is instead still inputting books from the cloned repo implementation. There is no functionality to download a package in our project right now.

Outcome of tests:



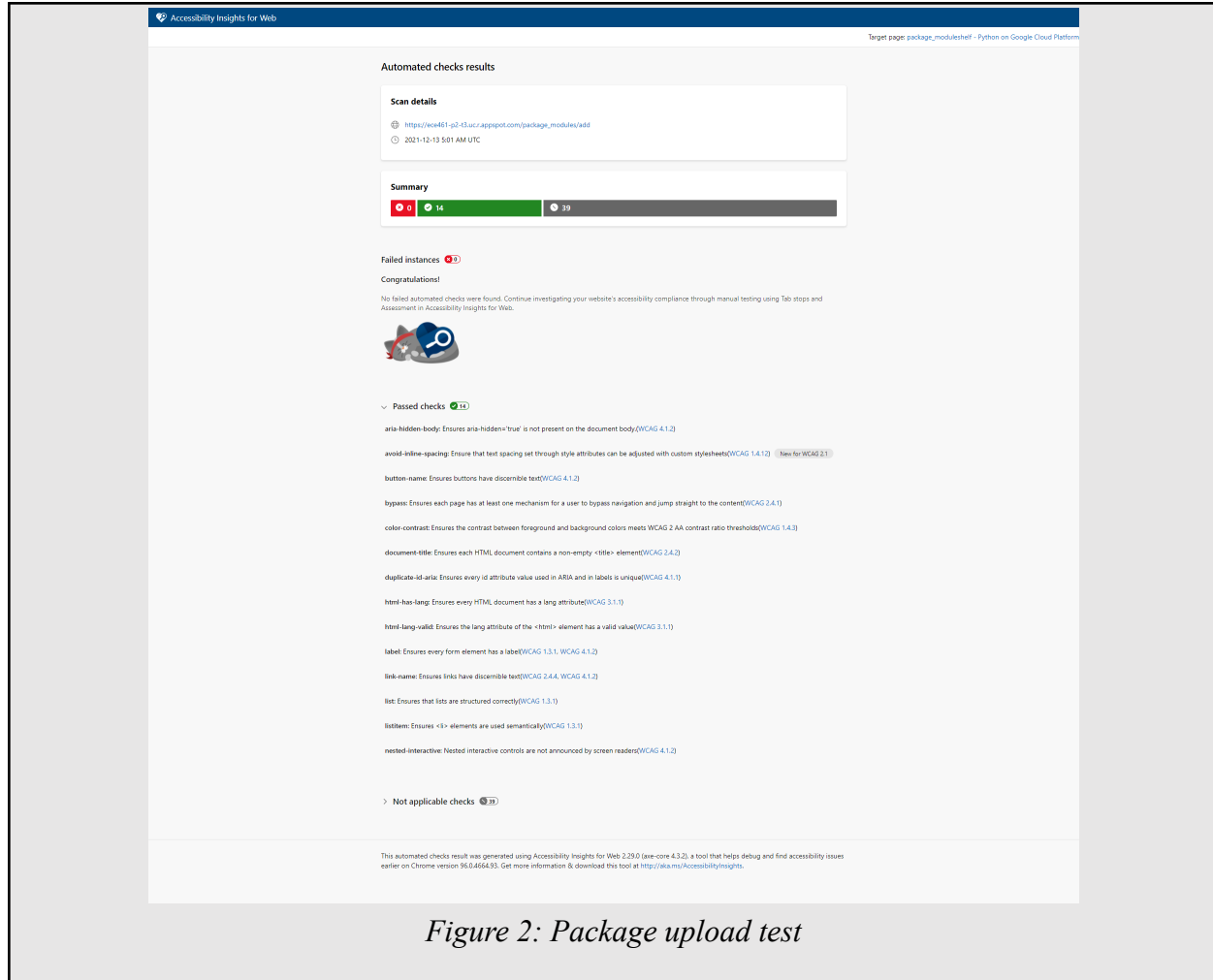
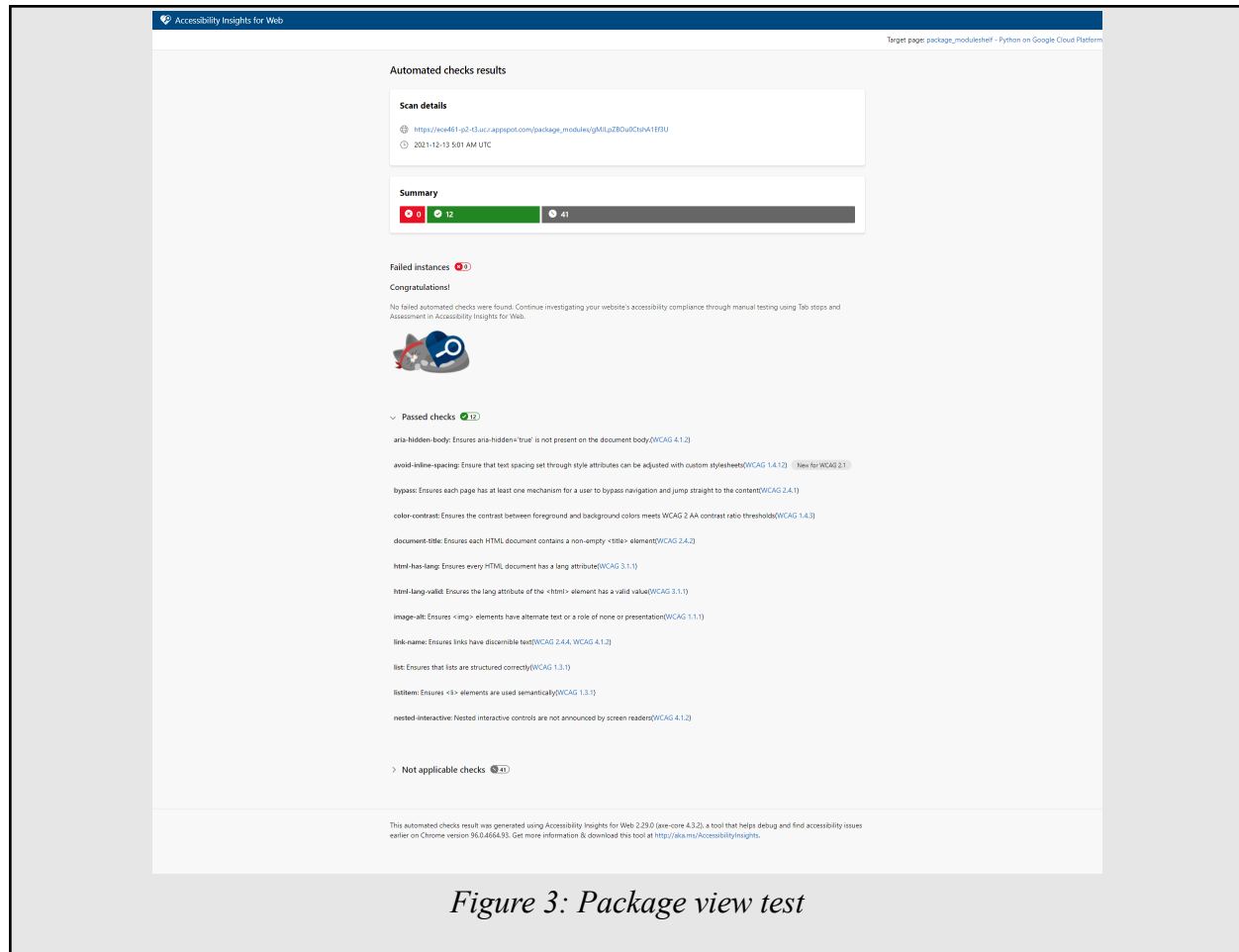


Figure 2: Package upload test



2. Describe any additional steps you took to consider accessibility (e.g. educational resources you consulted; design choices you made; implementation decisions that are not covered by Microsoft’s automated tests).

Looking at the tests above we found only color contrast errors and missing alt text image errors, which could be fixed quickly. Katie would have loved to implement more accessibility features, but the website was put aside to complete the API.

Non-functional requirements

Baseline

Security: STRIDE analysis

The terms used in this section are defined [here](#).

System model. Present one or more data-flow diagrams of your deployed system. (A whiteboard picture is fine, but use the correct symbols please).

- You may provide multiple DFDs to capture different aspects of the system.
- You may indicate multiple trust boundaries, e.g. for different classes of users.
- Each diagram should indicate at least the following entities: data flow; data store; process; trust boundary. You may include interactors and multi-process if needed.
- Each diagram should number the entities for reference later on.

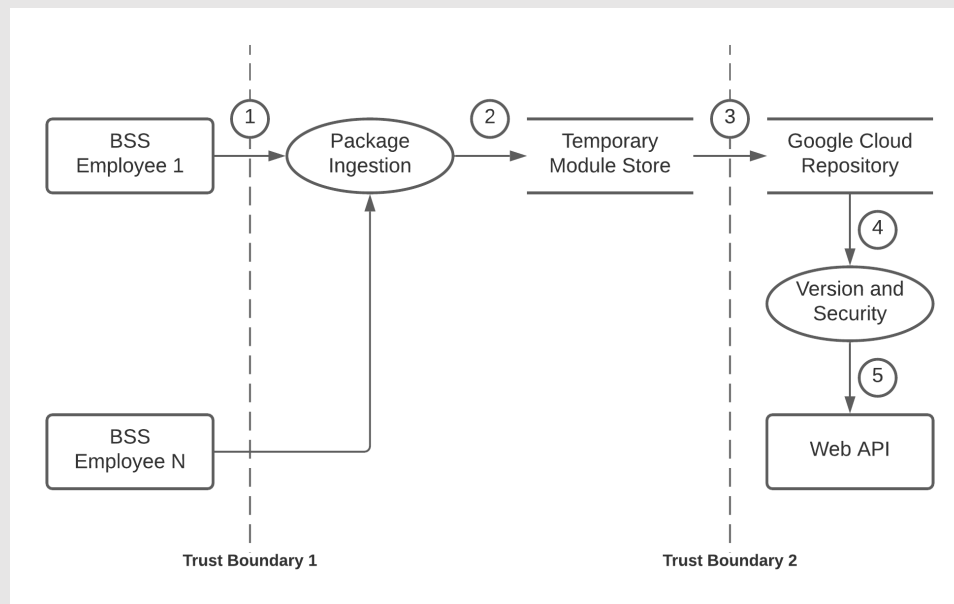


Figure 1 - Module Upload

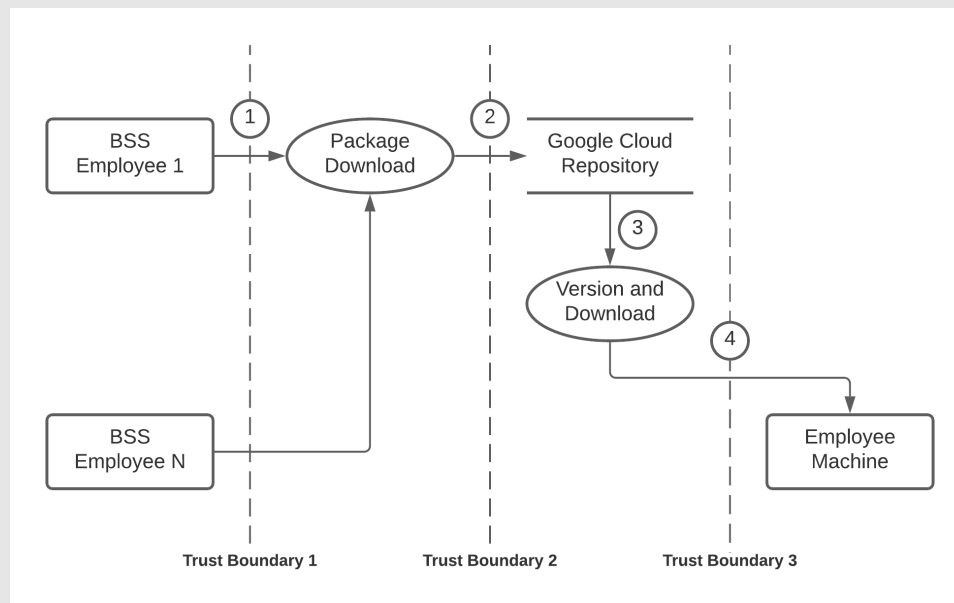


Figure 2 - Module Download

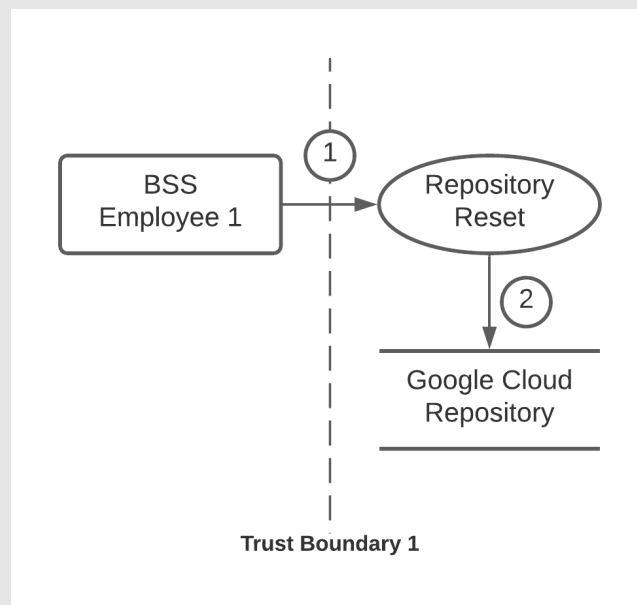


Figure 3 - Registry Reset

For each trust boundary indicated, describe the nature of the untrusted party involved (e.g. “outsider threat [e.g. external hacker]” or “insider threat [e.g. ACME employee with valid credentials]” or “infrastructure provider threat [GCP]”).

Figure 1 - Module Upload/Update

Trust Boundary 1

At the dataflow marked 1 in the figure, an employee of BSS is intended to submit a package for ingestion. The package data is moving from the employee’s computer to the project code. In this case a BSS employee, who has legitimate access to the system, may try to upload a package to the registry with malicious code in it. While the cloud platform shouldn’t run any uploaded packages, another employee might try to download and use the bad code.

Additionally, someone outside the company could try and pose as a BSS employee. They might try to upload tamper with packages like the previous example, or carry out a denial of service attack. This would prevent BSS from accessing trustworthy modules and could prevent company work from being completed.

Trust boundary 2

The dataflow marked 3 is between the temporary local storage of the package and the permanent storage on the Google Cloud Platform. A hacker on the same network as a BSS employee might try to tamper with the outgoing package. They might add malicious code or broken code that would impact BSS productivity.

Figure 2 - Module Download

Trust Boundary 1

The dataflow marked 1 is where a BSS employee tells the project to download a package from the GCP. In this case there isn't sensitive data being transferred, so the attack at this trust boundary would be a denial of service attack. Someone might pretend to be a BSS employee and flood the repository with download requests, preventing BSS from using the service.

Trust Boundary 2

The potential risk with dataflow 2 is similar to trust boundary 1. A hacker might attempt to make many download requests to the GCP without going through the API. This denial of service attack would harm BSS productivity.

Additionally, a hacker might try to access packages on the database they should not have access to. If there is proprietary code stored in the registry, there would be reason to try and download such code. This attack would give unauthorized users access to company materials without BSS knowing.

Trust Boundary 3

Dataflow 4 is at this trust boundary and is the final transfer of the downloaded package to the employee's machine. A hacker on the same network might try to intercept the transfer and gain access to company code that should not be accessible.

Figure 3 - Registry Reset

Trust Boundary 1

For this trust boundary, a BSS employee tells the GCP to erase the registry. A hacker might pretend to be a BSS employee to erase all the trustworthy modules used by the company. This would impact productivity because the modules would have to be reuploaded.

A malicious employee might also choose to reset the registry, in which case the same drop in productivity would occur.

Security requirements. The project document defines many requirements. Identify the security requirements of your system, aligned with the six security properties defined by the STRIDE article. These requirements may vary by system, depending on which features you implemented. It is possible that you will not have a requirement associated with every security property.

Confidentiality

- [All systems]: Observers on the network cannot directly observe client-server interactions.

- [GCP]: Google Cloud Armor rule set to only allow HTTPS connections making the data being transferred to the registry confidential.

Integrity

- [All systems]: During the continuous integration phase, the Bandit Python tool is used to scan for security issues. Systems were ensured to have no more severe than low priority issues.
- [GCP]: A user can request a package audit to ensure that the package they wish to download will not open their machine to security risks.

Availability

- [GCP]: A Google Cloud Armor load balancer was set up on the GCP to prevent DoS attacks.

Authentication

- None applicable

Authorization

- [Ingestion]: Users will not have access to the ingestion code so what qualifies as an ingestible package cannot be tampered with.

Nonrepudiation

- [GCP]: Each package is stored with update/upload history so a change to a package cannot be hidden.

Fill out this table for each [STRIDE property](#) (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege):

Stride property: Spoofing

Affected security properties: Authentication

Analysis of components:

- Figure 1, Component 1: BSS Employee
 - o Risk 1: Hacker may attempt to pose as an employee.
 - Mitigations applied:
 - o None.
 - Degree of risk resolution:
 - o None.
 - Suggestions for additional mitigations:
 - o Adding user authentication to ensure unauthorized users cannot make changes to the registry. This was an optional requirement we did not complete.

Stride property: Tampering

Affected security properties: Integrity

Analysis of components:

- Figure 1, Component 1: BSS Employee
 - o Risk 1: Malicious employee.
 - Mitigations applied:
 - o Ingestion code.
 - o Package audit.
 - Degree of risk resolution:
 - o With the two mitigations applied, an informed user of the system should be able to avoid the download and use of maliciously submitted bad code. If the ingestion step fails to block its upload, an employee can access the package audit to see if there are any potential security risks associated with the software.
- Figure 1, Component 3: Google Cloud Repository
 - o Risk 1: Hacker.
 - Mitigations applied:
 - o Package audit.
 - Degree of risk resolution:
 - o This mitigation works for a cautious user because it involves checking the audit before using a downloaded module.
 - Suggestions for additional mitigations:
 - o Additional scans run when downloading a package or within the GCP.
- Figure 3, Component 1: Repository Reset
 - o Risk 1: Malicious employee.
 - Mitigations applied:
 - o None.
 - Degree of risk resolution:
 - o None.
 - Suggestions for additional mitigations:
 - o A backup of the registry could be made daily so any loss of files could be reverted back to the day before.
 - o Use of authentication to allow only administrators to reset the registry. This was an optional requirement we did not complete.
 - o Risk 2: Hacker posing as employee.
 - Mitigations applied:
 - o None.
 - Degree of risk resolution:
 - o None.
 - Suggestions for additional mitigations:
 - o Same as the previous entry.

Stride property: Repudiation

Affected security properties: Nonrepudiation

Analysis of components:

- Figure 1, Component 1: BSS Employee
 - o Risk 1: Malicious employee.
 - Mitigations applied:
 - o Package history.
 - Degree of risk resolution:
 - o Every package is stored with data relating to any updates or changes made to it. This means that if an employee pushes a bad module to the GCP, they cannot later deny having done so.

Stride property: Information disclosure

- Figure 1, Component 2: Temporary Module Store
 - o Risk 1: Stolen Hardware.
 - Mitigations applied:
 - o Temporary module store is cleared after upload/download.
 - Degree of risk resolution:
 - o Clearing the temporary store of a module to upload ensures if hardware is misplaced, the uploaded package isn't available to access.
- Figure 2, Component 2: Google Cloud Repository
 - o Risk 1: Intercepted Packages.
 - Mitigations applied:
 - o Data is transferred to the GCP using the HTTPS protocol.
 - Degree of risk resolution:
 - o Packages intercepted while en route to the GCP should not be decipherable.
- Figure 2, Component 4: Employee Machine
 - o Risk 1: Intercepted Packages.
 - Mitigations applied:
 - o Same as the previous entry.
 - Degree of risk resolution:
 - o Same as the previous entry.

Stride property: Denial of service

Affected Security Properties: Availability

Analysis of Components:

- Figure 2, Component 1: Package Download
 - o Risk 1: Hacker DoS attack.
 - Mitigations applied:
 - o Google Cloud Armor
 - Degree of risk resolution:
 - o Google's cloud armor prevents DoS attacks on the GCP, so even if many download requests are made from the project, if an attack is detected a 403 return code will be given.
 - Suggestions for additional mitigations:

- Have a timeout on the client side that would prevent an individual machine from making too many requests at once, although this wouldn't stop DDoS attacks.
- Figure 2, Component 2: Google Cloud Repository
 - Risk 1: Hacker DoS attack.
 - Mitigations applied:
 - Same as the previous entry.
 - Degree of risk resolution:
 - Same as the previous entry.

Stride property: Elevation of privilege

Affected Security Properties: Authorization

Analysis of Components:

- None applicable.

Deployment: GCP

Complete the following table. It should include all GCP components that you used.

| Purpose | Selected GCP component(s) | Other GCP components considered | Justification for selected component |
|--|---|---------------------------------|--|
| Ensuring secure transfer of packages | Google Cloud default SSL policy on load balancer frontend | N/A | It's built in to GCP and provides access to a load balancer created by Google. |
| Content Delivery Network; geographical network showing content based on location | Google Cloud CDN on load balancer backend | Armor security policy | Armor requires some configuration, but CDN is enabled with a checkmark |
| deploying the app/API to GCP | Google Cloud App Engine | Kubernetes | This seemed the easiest to understand and was already implemented in the cloned repo we built from |
| storing the registry entries | Google FireBase/Firestore | N/A | This seemed the easiest to understand and was already implemented in the |

| | | | |
|--|------------------|-----|--|
| | | | cloned repo we built from |
| This is how APIs are defined inside an instance to be deployed | Google Endpoints | N/A | This is how APIs are defined inside an instance to be deployed |

Deployment: CI/CD

CI/CD: Provide prose and screenshot(s) demonstrating that your team is using GitHub actions to facilitate continuous integration (e.g. by running a *linter* and a *test suite* on every pull request) and continuous deployment.

You may reuse answers from Deliverable #1 as appropriate.

What steps is your team following prior to accepting a code change? (e.g. git-hooks, code review, linting, test suite, etc.)

Upon each push to the GitHub repository, three independent code testers are run on the Python code to ensure there are no outstanding problems with its implementation. These three testers are Pylint, Pyflakes, and Bandit. The output should be checked for any recommended fixes with urgent status before accepting code changes.

To ensure no security issues are present, the output from Bandit is checked for any issues with a severity rating higher than low. Pyflakes and Pylint don't have such rating systems so those two are checked manually before approval.

Provide a link to an example in your GitHub repo where your team followed this CI process:

<https://github.com/Purdue-ECE-461/project-2-3/commit/9f8db8800b6118dea3775f1a0759b6cbe1fd4fda#diff-10d4c749c752dffbc83d8c63aab905a374e4606a3c3a942ed653c95cdc7e309b>

How consistent have you been with this process? What is keeping you from full consistency?

We have been consistent in this process, however we have never had a scanner return any issue that would be cause for delaying the deployment. It has been helpful to see potential issues hidden in the code, but for so much of the software scanned, it was written by a third party and we were not inclined to make drastic changes to such software.

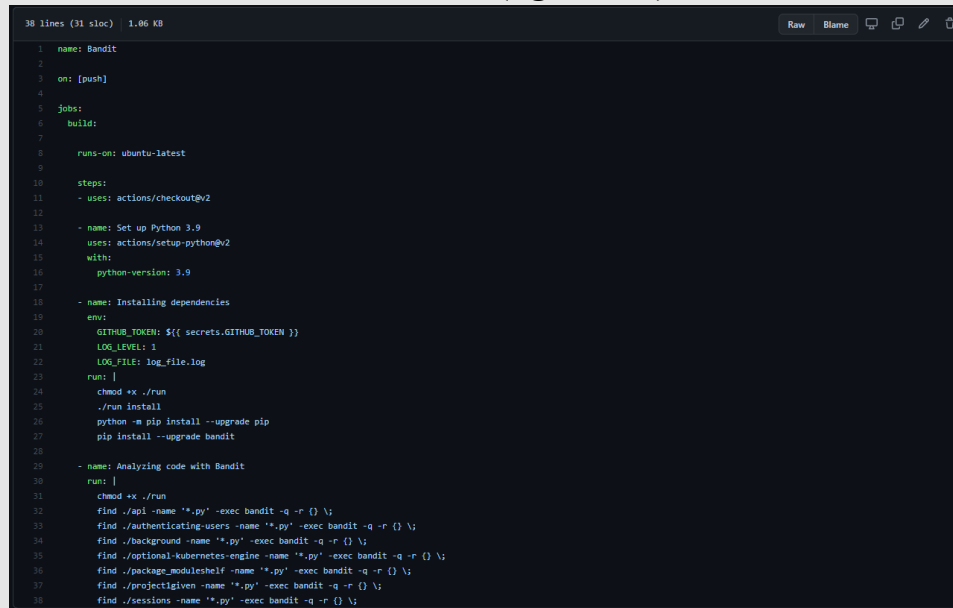
What aspects of your system are being tested automatically by your CI scheme?

All Python code that we did and did not write. GitHub Actions enters each folder containing Python in the repository and recursively scans its files and subdirectories. This means that our given implementation of Project 1 and the API implementation are being scanned in addition to template software (Google Cloud Platform code) from Google.

What kinds of defects might go uncaught, and how are you mitigating this risk?

The testers that we are using were chosen to try and address most kinds of defects. However there are still some things that might not be caught such as poor commenting or poor code quality. To help with this, we are doing our best to think about maintainability when pushing code. If the code is not maintainable, it should be reworked in order to fix that.

Additionally, files that are not written in Python might have security risks that the scanners we implemented cannot see. We should've addressed this during planning, however most of the code that is in the repository is written in Python. There are some Shell scripts but the majority were written by Google and we are choosing to trust this third party for this project.

Provide screenshots of the GitHub action file (e.g. YAML) that defines the CI stages


```

1  name: Bandit
2
3  on: [push]
4
5  jobs:
6    build:
7
8      runs-on: ubuntu-latest
9
10     steps:
11       - uses: actions/checkout@v2
12
13       - name: Set up Python 3.9
14         uses: actions/setup-python@v2
15         with:
16           python-version: 3.9
17
18       - name: Installing dependencies
19         env:
20           GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
21           LOG_LEVEL: 1
22           LOG_FILE: log_file.log
23         run: |
24           chmod +x ./run
25           ./run install
26           python -m pip install --upgrade pip
27           pip install --upgrade bandit
28
29       - name: Analyzing code with Bandit
30         run: |
31           chmod +x ./run
32           find ./api -name '*.py' -exec bandit -q -r {} \;
33           find ./authenticating-users -name '*.py' -exec bandit -q -r {} \;
34           find ./background -name '*.py' -exec bandit -q -r {} \;
35           find ./optional-kubernetes-engine -name '*.py' -exec bandit -q -r {} \;
36           find ./package_moduleshell -name '*.py' -exec bandit -q -r {} \;
37           find ./projectgiven -name '*.py' -exec bandit -q -r {} \;
38           find ./sessions -name '*.py' -exec bandit -q -r {} \;

```

Figure 1: bandit.yml

```

37 lines (30 sloc)  1.02 KB
1  name: Pyflakes
2
3  on: [push]
4
5  jobs:
6    build:
7
8      runs-on: ubuntu-latest
9
10     steps:
11       - uses: actions/checkout@v2
12
13       - name: Set up Python 3.9
14         uses: actions/setup-python@v2
15         with:
16           python-version: 3.9
17
18       - name: Installing dependencies
19         env:
20           GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
21           LOG_LEVEL: 1
22           LOG_FILE: log_file.log
23         run: |
24           chmod +x ./run
25           ./run install
26           python -m pip install --upgrade pip
27           pip install --upgrade pyflakes
28
29       - name: Analyzing code with Pyflakes
30         run: |
31           find ./api -name '*.py' -exec pyflakes {} \;
32           find ./authenticating-users -name '*.py' -exec pyflakes {} \;
33           find ./background -name '*.py' -exec pyflakes {} \;
34           find ./optional.kubernetes-engine -name '*.py' -exec pyflakes {} \;
35           find ./package_moduleshell -name '*.py' -exec pyflakes {} \;
36           find ./projectlliven -name '*.py' -exec pyflakes {} \;
37           find ./sessions -name '*.py' -exec pyflakes {} \;

```

Figure 2: pyflakes.yml

```

34 lines (30 sloc)  985 Bytes
1  name: Pylint
2
3  on: [push]
4
5  jobs:
6    build:
7
8      runs-on: ubuntu-latest
9
10     steps:
11       - uses: actions/checkout@v2
12
13       - name: Set up Python 3.9
14         uses: actions/setup-python@v2
15         with:
16           python-version: 3.9
17
18       - name: Install dependencies
19         env:
20           GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
21           LOG_LEVEL: 1
22           LOG_FILE: log_file.log
23         run: |
24           chmod +x ./run
25           ./run install
26           python -m pip install --upgrade pip
27           pip install pylint
28
29       - name: Analysing the code with pylint
30         run: |
31           find ./api -name '*.py' -exec pylint {} \;
32           find ./authenticating-users -name '*.py' -exec pylint {} \;
33           find ./background -name '*.py' -exec pylint {} \;
34           find ./optional.kubernetes-engine -name '*.py' -exec pylint {} \;
35           find ./package_moduleshell -name '*.py' -exec pylint {} \;
36           find ./projectlliven -name '*.py' -exec pylint {} \;
37           find ./sessions -name '*.py' -exec pylint {} \;

```

Figure 3: pylint.yml

Provide screenshot(s) of the test suite in action, e.g. the reports from the various tools you have configured, as run on one of your team's code changes.

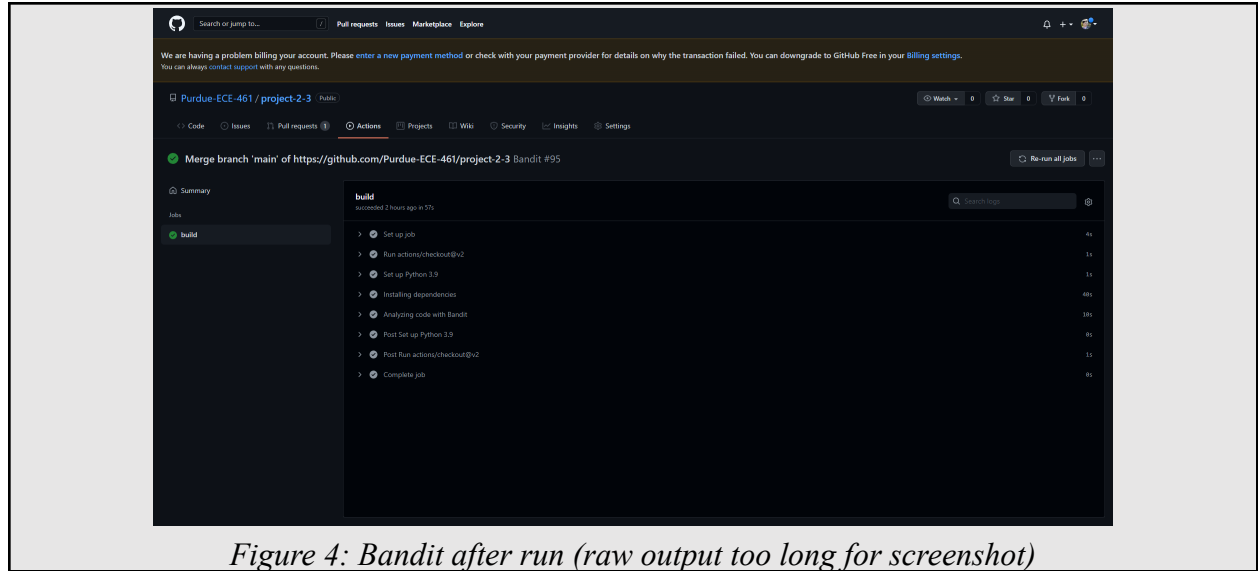
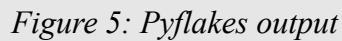


Figure 4: Bandit after run (raw output too long for screenshot)



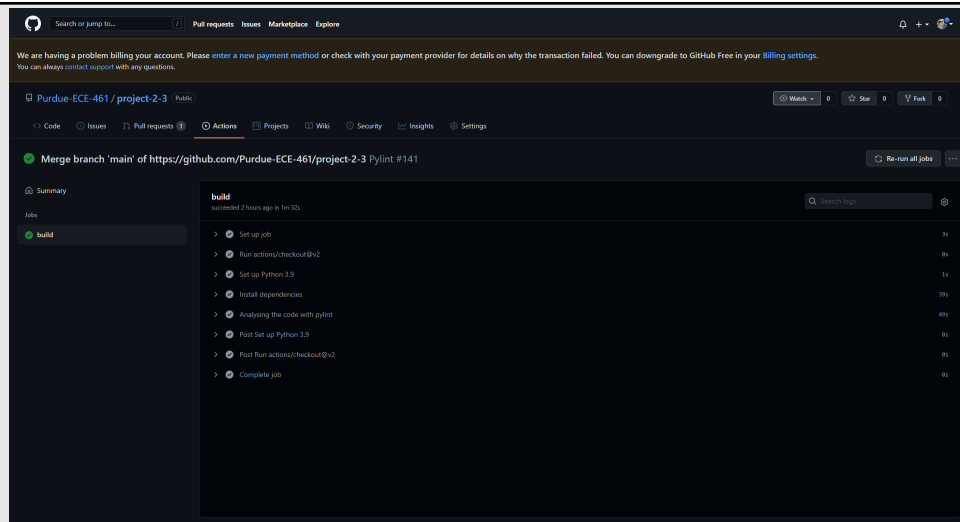


Figure 6: Pylint after run (raw output too long for screenshot)

Describe the extent to which you are able to “continuously deploy”. What is your team’s process to get your current prototype into a deployment on GCP?

Katie could use the `./run` file to deploy her local branch. Upon a successful push to the master branch, the system would be uploaded to the GCP. GitHub Actions, which was used for the continuous integration, was also used here. Uploading to GCP was handled within the GitHub Actions workflow in the `google.yml` file. The contents of this file are shown below:

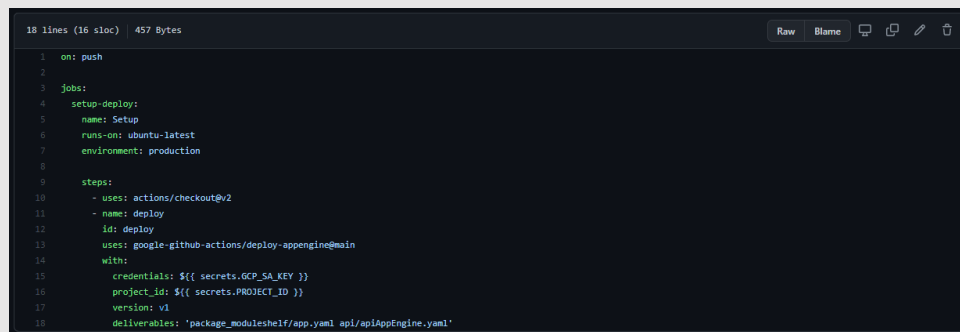


Figure 7: google.yml

Notes for the auto-grader

If your submission cannot be automatically parsed by the auto-grader described in the project specification, provide explanatory notes that the course staff can consider while scoring your submission. Be specific. Since this spec was provided well in advance, accommodating any deviations is at the discretion of the staff.

| Deviation | Details |
|-----------|---------|
|-----------|---------|

| | |
|-----|--|
| ... | |
| | |
| | |

Notes for the teamwork checks

In a typical team, each team member contributes a substantial amount of code. If on YOUR team, some team member contributed fewer than 100 lines of code, you may provide an explanation for us to consider.

Team member: Ashwin Senthilkumar

Important non-code contributions: This semester was quite challenging with juggling several projects including this one and senior design, as well as interviews for full-time jobs occurring nearly every week. As a result, I didn't have the time to contribute as much to this project as I would have hoped to, but I spent the time I had researching the technologies used in this project, like RESTful APIs, navigating GCP, and how a CI/CD pipeline is used to automate the software delivery process. I contributed to some of the semantic versioning metric calculation. I also worked on some of the documentation on the milestones during this project. While my overall code contribution was much less than I had planned, I was able to learn from my teammates on how the program works and what each requirement does.