



Phase 1:

The goal of phase 1 was to make a web application that takes code input from the user and outputs the compilation response. We can compile and output the result using docker. Using the Docker documentation we learned how to install Ubuntu, gcc and python onto our container. We then use node.js and shell.js to build the docker container and host on localhost:8080 off node.js.

The screenshot shows a web application titled "BreadGetter: Phase 1" in a dark-themed header. On the right side of the header, there is a "Language" dropdown menu and a "Run" button with a right-pointing arrow. Below the header, the main content area has an orange border. It contains two sections: "Enter Your Code:" with a large dark gray text area labeled "Place Code Here", and "Your Output:" with another large dark gray text area. Both text areas have small corner handles for resizing.

We are using bootstrap and local css to design a frontend for users to enter their code. On the navbar, we have a language dropdown and a run button. We have an input textarea and a readonly textarea for output. Since it was halloween month, our color scheme was spooks.

Running Python Code:


 BreadGetter: Phase 1 Python ▾ Run ►

Enter Your Code:

```
for i in range(1,10):  
    print(i)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Running C++ Code:

 BreadGetter: Phase 1 C++ ▾ Run ►

Enter Your Code:

```
#include <iostream>  
using namespace std;  
  
int main ()  
{  
    for (auto i = 0; i < 10; i ++){  
        cout << " i is: " << i << "\n";  
    }  
    return 0;  
}
```

```
i is: 0  
i is: 1  
i is: 2  
i is: 3  
i is: 4  
i is: 5  
i is: 6  
i is: 7  
i is: 8  
i is: 9
```

Phase 2:

The goal of phase 2 was to make a web application that simulates the pub/sub system. We made two sections to separate the publisher from the subscriber. The subscriber window has a topic dropdown and an enter name input. The publisher window has a topic dropdown and a

textarea for the publisher to input something about the topic. There is a read only output box that updates the viewer whenever there is a new subscriber, whenever there is new content from the publisher and if the subscriber receives the information. We use docker to run node.js and expose to port 8081.

BreadGetter Phase 2

Publishers

Select a topic to write about:

Choose a Topic

Input your text:

Message

Publish!

Subscribers

Enter Your Name:

John Doe

Select a topic to Subscribe to:

Choose a topic

Subscribe!

BreadGetter Phase 2

Publishers

Select a topic to write about:

Music

Input your text:

New album from Kanye West released today

Publish!

Subscribers

Enter Your Name:

manmeets

Select a topic to Subscribe to:

Music

Subscribe!

manmeets received message: New album from Kanye West released today
publishing message New album from Kanye West released today to topic music
new subscriber manmeets created with topic music

Phase 3:

Phase 3 was essentially phase 2 emulated three times. We made three separate instances of our phase 2 project, naming each one after a major city in the U.S (Buffalo, Houston, and San Francisco). To achieve the Pub/Sub emulation for each one of these instances and to allow ease of passing data between each one, each city has a publication file. When subscribers want to subscribe to a cities publication, they peak into the cities publication file to grab any new information. This publish/subscribe methodology lets subscribers from each city hear about information from other cities, and get notified when publishers publish to their respective city.