CLAIMS

What is claimed is:

1. A computer system comprising:

a user computer comprising a first processor coupled to one or more first non-transitory computer-readable storage media storing a user pipeline automation script comprising one or more first sequences of instructions specifying one or more API calls to an application programming interface (API), and a user application, wherein the user computer is coupled directly or indirectly via one or more networks to a script automation processor and a cloud computing service;

a virtual compute instance and a virtual storage instance associated with one or more second non-transitory computer-readable storage media storing one or more second sequences of instructions defining an API implementation of the API, a graph server, and one or more programming language runtime interpreters, and which, when executed using the virtual compute instance, cause the virtual compute instance to execute:

executing the script automation processor;

using the script automation processor, interpreting the user pipeline automation script using a particular programming language runtime interpreter among the one or more programming language runtime interpreters and executing code of the API implementation corresponding to the one or more API calls of the user pipeline automation script to interoperate with a container build utility hosted using the virtual compute instance to create and store one or more programmatic containers in memory of the script automation processor, the containers corresponding to functions that the user pipeline automation script and the user application invoke and created as needed to instantiate and execute the user application and user pipeline automation script, including any dependencies;

creating and storing a directed acyclic graph (DAG) in the memory of the script processor, the DAG comprising nodes and edges corresponding to the dependencies of the containers;

after interpreting the user pipeline automation script to automatically build, test, or deploy the user application in the cloud computing service, writing one or more build artifacts to a specified host location.

2. The computer system of claim 1, the script automation processor further comprising the second sequences of instructions which, when executed using the virtual

compute instance, cause the virtual compute instance to execute the running by executing one or more GraphQL queries against the API implementation.

3. The computer system of claim 1, the script automation processor further comprising two or more programming language runtime interpreters, wherein each programming language runtime interpreter among the two or more programming language runtime interpreters is programmed to interpret a different programming language used in the user pipeline automation script.

4. The computer system of claim 1, wherein the user pipeline automation script further comprises the first sequences of instructions specifying one or more API extensions;

wherein the script automation processor further comprises the second sequences of instructions which, when executed using the virtual compute instance, cause the virtual compute instance to execute:

installing the one or more API extensions in association with the API implementation;

during the interpreting, based on the user pipeline automation script, invoking one or more of the one or more API extensions.

5. The computer system of claim 4, further comprising executing each of the one or more API extensions in a sandboxed region of the main memory of the script automation processor.

6. The computer system of claim 4, each of the one or more API extensions comprising sequences of instructions that are executable as part of the API implementation to interoperate with any one or more of Yarm, Netlify, Git, OCI, Vecel, and Alpine Linux.

7. The computer system of claim 1, wherein the user pipeline automation script further comprises a workflow configuration in a markup language.

8. The computer system of claim 1, wherein the user pipeline automation script further comprises at least one reference to a software development kit (SDK), and wherein the script automation processor comprises an SDK interface responsive to function invocations via the at least one reference.

9. The computer system of claim 1, wherein the SDK is programmed to run one or more of Go, Python, Javascript, Typescript, and Bash to generate one or more client stubs of a build workflow.

10. A computer-implemented method comprising:

using a script automation processor that is hosted using a virtual compute instance and a virtual storage instance associated with one or more second non-transitory computer-readable storage media storing one or more second sequences of instructions defining an API implementation of the API, a graph server, and one or more programming language runtime interpreters, receiving a user pipeline automation script comprising one or more first sequences of instructions specifying one or more API calls to an application programming interface (API), and a user application;

using the script automation processor, interpreting the user pipeline automation script using a particular programming language runtime interpreter among the one or more programming language runtime interpreters and executing code of the API implementation corresponding to the one or more API calls of the user pipeline automation script to interoperate with a container build utility hosted using the virtual compute instance to create and store one or more programmatic containers in memory of the script automation processor, the containers corresponding to functions that the user pipeline automation script and the user application invoke and created as needed to instantiate and execute the user application and user pipeline automation script, including any dependencies;

using the script automation processor, creating and storing a directed acyclic graph (DAG) in the memory of the script processor, the DAG comprising nodes and edges corresponding to dependencies of the containers;

using the script automation processor, after interpreting the user pipeline automation script to automatically build, test, or deploy the user application in a cloud computing service, writing one or more build artifacts to a specified host location.

11. The computer-implemented method of claim 10, further comprising, using the script automation processor, running the user pipeline automation script by executing one or more GraphQL queries against the API implementation.

12. The computer-implemented method of claim 10, the script automation processor further comprising two or more programming language runtime interpreters, wherein

each programming language runtime interpreter among the two or more programming language runtime interpreters is programmed to interpret a different programming language used in the user pipeline automation script.

13. The computer-implemented method of claim 10, wherein the user pipeline automation script further comprises the first sequences of instructions specifying one or more API extensions, the method further comprising, using the script automation processor, installing the one or more API extensions in association with the API implementation, and during the interpreting, based on the user pipeline automation script, invoking one or more of the one or more API extensions.

14. The computer-implemented method of claim 13, further comprising executing each of the one or more API extensions in a sandboxed region of the main memory of the script automation processor.

15. The computer-implemented method of claim 13, each of the one or more API extensions comprising sequences of instructions that are executable as part of the API implementation to interoperate with any one or more of Yarm, Netlify, Git, OCI, Vecel, and Alpine Linux.

16. The computer-implemented method of claim 10, wherein the user pipeline automation script further comprises a workflow configuration in a markup language.

17. The computer-implemented method of claim 10, wherein the user pipeline automation script further comprises at least one reference to a software development kit (SDK), and wherein the script automation processor comprises an SDK interface responsive to function invocations via the at least one reference.

18. The computer-implemented method of claim 10, wherein the SDK is programmed to run one or more of Go, Python, Javascript, Typescript, and Bash to generate one or more client stubs of a build workflow.

19. One or more non-transitory computer-readable storage media storing one or more sequences of instructions which, when executed using one or more processors, cause the one or more processors to:

using a script automation processor that is hosted using a virtual compute instance and a virtual storage instance associated with one or more second non-transitory computer-readable storage media storing one or more second sequences of instructions defining an API implementation of the API, a graph server, and one or more programming language runtime interpreters, receiving a user pipeline automation script comprising one or more first sequences of instructions specifying one or more API calls to an application programming interface (API), and a user application;

using the script automation processor, interpreting the user pipeline automation script using a particular programming language runtime interpreter among the one or more programming language runtime interpreters and executing code of the API implementation corresponding to the one or more API calls of the user pipeline automation script to interoperate with a container build utility hosted using the virtual compute instance to create and store one or more programmatic containers in memory of the script automation processor, the containers corresponding to functions that the user pipeline automation script and the user application invoke and created as needed to instantiate and execute the user application and user pipeline automation script, including any dependencies;

using the script automation processor, creating and storing a directed acyclic graph (DAG) in the memory of the script processor, the DAG comprising nodes and edges corresponding to dependencies of the containers;

using the script automation processor, after interpreting the user pipeline automation script to automatically build, test, or deploy the user application in a cloud computing service, writing one or more build artifacts to a specified host location.

20. The non-transitory computer-readable storage media of claim 19, further comprising sequences of instructions which when executed cause running the user pipeline automation script by executing one or more GraphQL queries against the API implementation.

21. The non-transitory computer-readable storage media of claim 19, the script automation processor further comprising two or more programming language runtime interpreters, wherein each programming language runtime interpreter among the two or more

programming language runtime interpreters is programmed to interpret a different programming language used in the user pipeline automation script.

22. The non-transitory computer-readable storage media of claim 19, wherein the user pipeline automation script further comprises the first sequences of instructions specifying one or more API extensions, the non-transitory computer-readable storage media further sequences of instructions which when executed cause installing the one or more API extensions in association with the API implementation, and during the interpreting, based on the user pipeline automation script, invoking one or more of the one or more API extensions as part of automatically building, testing, or deploying the user application in the cloud computing service.

23. The non-transitory computer-readable storage media of claim 22, further comprising sequences of instructions which when executed cause executing each of the one or more API extensions in a sandboxed region of the main memory of the script automation processor.

24. The non-transitory computer-readable storage media of claim 22, each of the one or more API extensions comprising sequences of instructions that are executable as part of the API implementation to interoperate with any one or more of Yarm, Netlify, Git, OCI, Vecel, and Alpine Linux.

25. The non-transitory computer-readable storage media of claim 19, wherein the user pipeline automation script further comprises a workflow configuration in a markup language.

26. The non-transitory computer-readable storage media of claim 19, wherein the user pipeline automation script further comprises at least one reference to a software development kit (SDK), and wherein the script automation processor comprises an SDK interface responsive to function invocations via the at least one reference.

27. The non-transitory computer-readable storage media of claim 19, wherein the SDK is programmed to run one or more of Go, Typescript, and Bash to generate one or more client stubs of a build workflow.