Building Tapis v3 Streams API Support for Real-Time Streaming Data Event-Driven Workflows

Smruti Padhy
Texas Advanced Computing Center
Austin, TX, USA
spadhy@tacc.utexas.edu

Anagha Jamthe
Texas Advanced Computing Center
Austin, TX, USA
ajamthe@tacc.utexas.edu

Sean B. Cleveland
University of Hawaii - System
Honolulu, HI, USA
seanbc@hawaii.edu

Jack A. Smith
Center for Environmental, Geotechnical, and Applied Sciences
Marshall University
Austin, TX, USA
smith1106@marshall.edu

Joe Stubbs
Texas Advanced Computing Center
Austin, TX, USA
jstubbs@tacc.texas.edu

Christian Garcia
Texas Advanced Computing Center
Austin, TX, USA
cgarcia@tacc.utexas.edu

Michael Packard
Texas Advanced Computing Center
Austin, TX, USA
mpackard@tacc.texas.edu

Steve Terry
Texas Advanced Computing Center
Austin, TX, USA
sterry@tacc.texas.edu

Julia Looney
Texas Advanced Computing Center
Austin, TX, USA
jlooney@tacc.texas.edu

Richard Cardone
Texas Advanced Computing Center
Austin, TX, USA
rcardone@tacc.texas.edu

Maytal Dahan
Texas Advanced Computing Center
Austin, TX, USA
maytal@tacc.utexas.edu

Gwen A. Jacobs
University of Hawaii - System
Honolulu, Hawaii, USA
gwenj@hawaii.edu

ABSTRACT

The Tapis framework, an NSF-funded project, is an open-source, scalable API platform that enables researchers to perform distributed computational experiments securely and achieve faster scientific results with increased reproducibility. Tapis Streams API focuses on supporting scientific use cases that require working with real-time sensor data. The Streams Service, built on the top of the CHORDS time-series data service, allows storing, processing, annotating, querying, and archiving time-series data. This paper focuses on the new Tapis Streams API functionality that enables researchers to design and execute real-time data-driven event workflow for their research. We describe the architecture and design choices towards achieving this new capability with Streams API. Specifically, we demonstrate the integration of Streams API with Kapacitor, a native data processing engine for time-series database InfluxDB, and Abaco, an NSF Funded project, web service, and distributed computing platform providing function-as-a-Service (FaaS). The Streams API, which includes a wrapper interface for the Kapacitor alerting system, can define and enable alerts. Finally, simulation results from the water-quality use case depict that Streams API's new capabilities can support real-time streaming data event-driven workflows.

PEARC '21, July 18–22, 2021, Boston, MA, USA

ACM ISBN 978-1-4503-8292-2/21/07…$15.00
https://doi.org/10.1145/3437359.3465567

CCS CONCEPTS

• **Information systems → Computing platforms**.

KEYWORDS

Alerts, CHORDS, Event-driven workflow, Kapacitor, Real-time, Tapis, Time-series data

ACM Reference Format:
Smruti Padhy, Anagha Jamthe, Sean B. Cleveland, Jack A. Smith, Joe Stubbs, Christian Garcia, Michael Packard, Steve Terry, Julia Looney, Richard Cardone, Maytal Dahan, and Gwen A. Jacobs. 2021. Building Tapis v3 Streams API Support for Real-Time Streaming Data Event-Driven Workflows. In *Practice and Experience in Advanced Research Computing (PEARC '21), July 18–22, 2021, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3437359.3465567

1 INTRODUCTION

The rise of inexpensive devices in the Internet of Things (IoT), instruments, and sensors to observe and measure everything in recent years have led to a deluge of data and demand for support related to storing, processing, and analyzing time-series data. Further, as more Machine Learning and Artificial Intelligence systems are developed and come online, the need for continuous learning data and datasets that are leveraged for training and used as input for driving event-triggered computation is increasing [22]. Many science use cases based on monitoring require ongoing data processing or special processing/modeling and notification of anomalous or

special events that can be identified by processing and computing the data as it arrives. However, many of the current computational infrastructure and systems built around computation are designed for one-off or batch computation of longitudinal datasets. Systems that have been designed for sensors are often aimed at industry, and either are complex to deploy and maintain or, if hosted, are expensive. This leaves a gap for hosted academic streaming data solutions capable of supporting data event-driven computational workflows.

To help address this cyberinfrastructure need to support streaming data for science, the Texas Advanced Computing Center (TACC) and the University of Hawaii (UH) have developed an open source unified middleware API infrastructure platform, Tapis [7, 21], with collaborative features to fill gaps in the existing streaming time-series data landscape. Tapis is accessible as a hosted platform as a service (PaaS) or for on-site deployments where more local control is necessary.

The middleware that supports streaming data and new multi-stream and multi-variable data event-driven workflows features is known as the Tapis Streams API (Figure 1) and has been designed to enable several core features:

- Data Ingestion - Support for time-series float and string data from devices and instruments.
- Data Annotation - integrated extendable annotation for data curation and discovery.
- Data Storage - hosted two-tiered storage for fast windowed processing and long-term storage within InfluxDB and MongoDB.
- Data Archival - support archival and data replication within external CHORDS instances and flat file formats.
- Data Retrieval (with temporal and spatial support) - search APIs that enable spatial extent and temporal constraints for advanced queries.
- Data-driven Event Workflows - support for defining evaluation functions on multiple incoming data streams and evaluating multiple measurement variables to trigger analysis workflows with tight integration into the Tapis Abaco [20] (Function as a service) and computational Applications and Jobs services (Figure 1).
- Collaboration Support - role-based access allowing multiple users to share and manage both data and workflows.

Previous work has focused on the first stage of handling the streaming data life-cycle by supporting ingestion, storage, management/curation and serving streaming data with the Tapis Streams API [18].

In this paper, we present features that pave the way to automated data event-driven workflows that can leverage many of the cutting edge analysis libraries like Apache Spark-Streaming [15] or the many deep learning frameworks through Abaco or the Tapis Applications and Jobs service. Further, we will discuss the background in Section 2, a real-time event-driven example workflow in Section 3, design and implementation of the new APIs in Section 4, experimental evaluation in Section 5, and opportunities related to the Tapis Streams API in Section 6.

2 BACKGROUND

2.1 CHORDS

CHORDS [19] is a cloud-hosted real-time data service infrastructure that provides a graphical interface for acquiring, storing, and analyzing data streams via cloud services and the Internet. The CHORDS portal is a Ruby on Rails [12] web application, which leverages databases (InfluxDB and MySQL [17]) that accepts real-time data from distributed instruments and serves the measurements to anyone on the Internet. The basic data model supported by CHORDS is Sites, which is a physical geographical location; an instrument hosted at the site has one or more variables/sensors, which generate real-time streaming data, also known as measurements. Metadata about sites, instruments, and variables are stored in MySQL database, whereas measurements posted to CHORDS are written to InfluxDB using HTTP requests. Scientists and analysts can easily fetch the data in real-time from the CHORDS portal, delivered directly to browsers, programs, and mobile apps.

2.2 Tapis V3 Streams API

Tapis V3 Streams API has been developed by researchers at UH and TACC to support the integration of streaming data workflows, storage, retrieval and analysis of time and location-sensitive sensor data. The API leverages Python Flask [5] web framework to conform to other Tapis services. Streams API provides a production-quality service that builds on top of the CHORDS project for real-time data services. It extends the CHORDS primary data models, including site, instrument, and variable, with additional metadata, including adding spatial indexing and permissions. Streams API's design conforms to the OpenAPI 3.0 specification. A live-doc describing all the REST endpoints is available for viewing [13]. Streams microservice leverages other Tapis microservices such as Security Kernel, Metadata, Tokens to provide a secure, scalable and reproducible service. Every user request first goes through the Security Kernel for authorization and authentication check to ensure that the user has the necessary role to perform requested action on Streams resources. Tokens service provides a signed service JWT, which lets the Security Kernel and Metadata service to know that request is coming from an authentic source, i.e., Streams service. Metadata service provides a backend database for Streams API, which stores all the metadata associated with the Streams resources.

2.3 Tapis V3 Streams Resources - Projects, Sites, Instruments, Variables, Measurements

Resources in Streams API are created in hierarchical order, which allows for better management and access control. Projects resources are at the top level of the hierarchy, followed by sites, instruments, variables, and measurements, respectively. A project is a logical grouping of one or more site(s). A Site is a geographical location with geospatial coordinates of latitude, longitude, and elevation, which hosts one or more instruments. An instrument can have multiple embedded sensors, which generate real-time measurements that can be stored in the time-series InfluxDB and retrieved for analyses. At the time of project creation, project metadata such as the principal investigator, project URL, funding resource, etc. is stored in the back-end MongoDB collection. A list of authorized

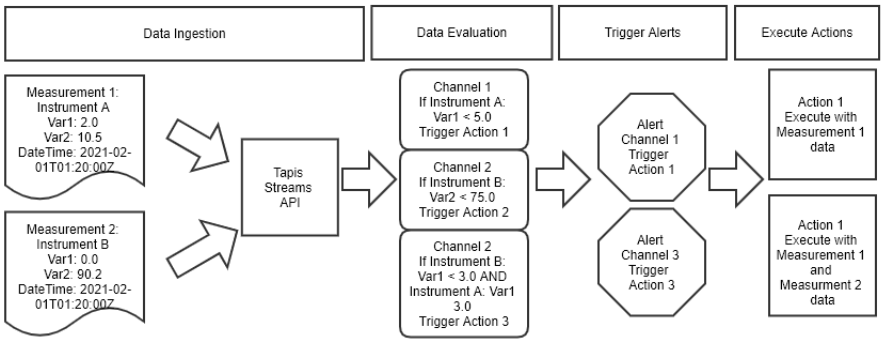

Figure 1: An example of the new multi-stream and multi-variable data event-driven streaming data workflow features in the Tapis Streams API illustrating data streaming from two instruments (A and B) and being evaluated by three conditions. Each Channel defines a set of conditions to evaluate data as it is being ingested that results in two Alerts that trigger actions. These actions could be third party API calls, Tapis Abaco Actors, or Tapis jobs.

users can be added to various project roles to have controlled access to the project resources. User roles such as "admin", "manager", and "user" are set up in the security kernel to decide if the user can perform CRUD operations on the stream's resources. Every request to access the project resource or documents within (i.e., sites, instruments, variables) goes through a security kernel check, and only the authorized user requests are allowed to be processed. Streaming API components necessary for event-driven workflow are discussed in detail in Section 4.

2.4 InfluxDB

InfluxDB, an opensource time-series database, provides high throughput data ingestion, compression, and real-time querying capabilities [9]. Its ability to handle millions of data points per second makes it extremely useful for storing and processing real-time sensor data. Streams API leverages InfluxDB 1.x for storing and retrieving measurement values generated by sensors embedded in the instruments. Measurements stored have a creation timestamp. InfluxDB 1.x provides native support for Kapacitor [11], which is used for real-time data processing, whereas the latest InfluxDB 2.x version has the Kapacitor component integrated. With InfluxDB 1.x, Kapacitor subscribes to InfluxDB and receives a copy of all data points, as they are written [10].

2.5 Kapacitor

Kapacitor is a native real-time streaming data processing engine for InfluxDB 1.x. It can process batch and real-time streaming data from InfluxDB by defining a monitoring and processing task to perform on the in-coming data via its programming language TICKscript [14]. Kapacitor provides ways to write a TICKscript template task, define a task using the template, and set up alerts and trigger actions. Since Streams API uses InfluxDB for storing and retrieving time-series data, integration with Kapacitor becomes a default choice towards achieving real-time data event-driven workflows.

2.6 Abaco

Abaco (Actor Based Containers) is an open-source distributed computing platform, and web-based Application Programming (API) hosted at TACC that enables clients/users to compute atomic, independent workloads or functions on cloud infrastructure [20]. It uses Linux container technology and actor model to provide Function-as-a-Service. It was prototyped in 2015 and got NSF funding in 2017. Since then, it has been used by several research projects. Approximately 50,000 actors have been registered, and nearly 1 million executions have been performed with a total runtime of 25 million seconds. Abaco implements the Actor Model where functions (referred to as "actors") execute in response to receiving messages. The functions run with an authenticated context that allows them to make requests to other Tapis APIs to perform actions such as data transfer or job submission. Towards supporting event-driven workflows for real-time sensor data, the Streams API internals generates a message describing the alert event, including details such as the measurements and thresholds exceeded, and it sends the message to the actor registered. In this way, the actor can take arbitrary actions in response to the event, including but not limited to performing local analysis or submitting jobs to HPC clusters.

2.7 Kubernetes Deployment

The Tapis project includes official tooling to ease deployment and administration of all Tapis API services and components, allowing institutions to run a subset of the services on-premise while utilizing the primary deployment at TACC for other services. This official tooling targets the Kubernetes [16] container orchestration system, which provides a number of high-level features, including networking and service discovery, container scheduling and scaling, persistent storage, monitoring, and self-healing. The Streams API deployment consists of five primary components: the Python API, the CHORDS server, the Kapacitor process, and two databases: InfluxDB for time-series measurements and MySQL which CHORDS uses. Configuration of the deployments utilizes Kubernetes configmaps and secrets objects. A single startup script within the official deployment tools will create the necessary configmaps and

secrets, persistent volume claims and deployments to start the entire Streams API stack. For more details on the architecture, see Section 4.4.

3 REAL-TIME EVENT-DRIVEN WORKFLOW EXAMPLE

Researchers at Marshall University currently leverage 3 NSF-funded projects: Appalachian Freshwater Initiative (AFI) [2], Sensing and Educating the Nexus to Sustain Ecosystems (SENSE) [3] and Extensible Geospatial Data Framework towards FAIR Science (GeoEDF) [4] to address the water quality of streams in Appalachia. They propose to benefit from having real-time analysis connected to streaming data through a service like TAPIS and its Streaming API integrated into their Science Gateway. This use case involves the collection and sharing of water quality data using the EPA's WQP/WQX APIs and a local staging service call WQBase for discrete data and an instance of CHORDS for continuously-monitored (real-time) sensor data, with the need for on-the-fly processing, potentially on remote HPC resources. The goal of integrating Tapis Streams with their water quality science gateway is to use the Streams Channels and Alerts APIs to detect anomalies in the measured data, correct for calibration errors, generate alerts when certain thresholds are exceeded, compute rolling statistics,compare or combine with other data sources, use machine learning and image processing to recognize conditions favorable to harmful algae bloom (HAB) formation, and to push summary data to WQBase for staging, pending further vetting and analysis before publishing to WQP via WQX.

4 TAPIS V3 STREAMS REST API

Tapis V3 Streams have REST APIs for streaming resources: projects, sites, instruments, variables, for storing, processing, and querying time-series data [18]. The current work adds two new REST API implementations for Templates and Channels. These APIs enable researchers to define and execute their scientific streaming data workflow. This section focuses on the implementation and architecture of these new capabilities and their role in supporting the streaming data event-driven workflow. In the subsequent subsections, we describe Tasks, Templates, and Channels APIs.

4.1 Task and Template

A task is a work to be performed on a streaming dataset. A template is a re-usable definition that represents a task. A typical task's structure encompasses monitoring the incoming streaming data or measurements, aggregating values, raising alerts when the measurement values satisfy specified boolean conditions, and sending the critical data that triggered the alert to perform specified actions. We alternatively refer to a template as a template task, which is used to define a channel. The notion of task and template in Streams service corresponds to the task and template in Kapacitor. The Streams API wraps Kapacitor's templates API to provide support for templates in the Streams service. We will define Channel in Section 4.2.

4.1.1 Templates REST API. Table 1 shows the Templates REST API to create, update, list, and delete templates. A Streams API user can create template by making a HTTP POST request to the /v3/streams/templates endpoint, including a JSON formatted request body containing the template information. The request requires three fields - *template_id*, *type*, and *script*, where *template_id* is a unique identifier for the template to be created, *type* represents data type and is set to *stream* for streaming data, and *script* is a TICKScript describing the structure of the task. We refer to this script as template henceforth as it is the actual template task.

Table 1: Streams Templates REST API

Method	REST API	Description
GET	/v3/streams/templates	Lists templates
POST	/v3/streams/templates	Creates a Template
GET	/v3/streams/templates/{id}	Get details of the specified Template
PUT	/v3/streams/templates/{id}	Update the specified Template definition
DEL	/v3/streams/templates/{id}	Delete the specified Template

```
1  var measurement string
2  var channel_id string
3  var crit1 lambda
4  var crit2 lambda
5  var crit3  lambda
6
7  var var1 = stream
8  |from()
9    .measurement(measurement)
10   .where(lambda:crit1)
11
12 var var2 = stream
13 |from()
14   .measurement(measurement)
15   .where(lambda:crit2)
16
17 var1
18   |join(var2)
19     .as('var1','var2')
20     .tolerance(1ms)
21     .fill(0.0)
22     .streamName('multic')
23   |alert()
24     .id(channel_id +  ' {{ .Name }}/{{ .Group }}/{{.TaskName}}')
25     .crit(lambda:crit3)
26     .noRecoveries()
27     .message('{{.ID}} is {{ .Level}} at time: {{.Time}} as value
   exceeded the threshold')
28     .details('')
29     .post()
30     .endpoint('api-alert')
31     .captureResponse()
32 _ |httpOut('msg')
```

Figure 2: A TICKScript template for two variables

4.1.2 Template Design. A template is a TICKScript defining a task with variables that are set during the Channel definition. Figure 2 shows an example of a template representing a template task that monitors two Streams resource variables values and raises an alert if a specific boolean condition is satisfied. A template in the Streams service has three critical sections:

- **Declaration of TICKScript variables**. These declarations are required for the criteria to filter incoming streaming data and for the boolean conditions specified in the Channel definitions. Line numbers 1-5 in Figure 2 show the set of TICKScript variables defined for the template, which get assigned values from the Channel definition during channel/task creation.
- **Setting up TICKScript variables to filter data streams**: Variables *var1* and *var2* defined on lines 7-15 in the example template shown in Figure 2, represent Streams resource variables (sensors) measurement values. *crit1* and *crit2* represent the boolean conditions to filter the data points from the measurement tables. Note that there could be several Streams resource variables for an instrument. In the above

template script, we monitor only two variables and correspondingly filter the data points from the measurement table. As leveraged in the Streams service, the CHORDS data model uses the schema (*timestamp*, *inst*, *site*, *test*, *value*, *var*) for storing measurements in the InfluxDB. The *timestamp* field represents the date-time of measurement creation in an UTC standard time, *site* is the associated CHORDS site id, *inst* denotes CHORDS instrument id, *var* represents CHORDS variable(sensor) id, and the *value* contains the actual measurement value generated by the variable. Two variables in the example represent two data points in the measurement table, i.e., two rows of the measurement table. To monitor two variables simultaneously and check the criteria, we need to join/aggregate them as a single stream based on their timestamp. Once $var1$ and $var2$ are set with the filtered data points streams by applying *crit1* and *crit2*, respectively, these data are joined/ paired based on timestamps within a tolerance interval as shown in line numbers 17-22. In the example, the tolerance interval is 1 ms. The joining of data streams indicates that these data arrive within the time interval 1 ms and are close enough to be considered as a single stream and based on which alerts can be defined.

- **Alerts Definition**: An alert is defined after joining the streams of data points for more than one variable. An alert definition requires a boolean condition, an alert level, an alert message, an option to send details of the data point raising an alert, and a destination end-point where the alerts are posted. In the example template line numbers 23-31, *crit3* is a boolean condition on *var1* and *var2*, and *'api-alert'* in the destination end-point where the alert is posted. For each tenant, separate end-points are configured in the configuration file kapacitor.conf.

Another key aspects of the Template is the representation of a boolean conditional expression for the variables. We leveraged the Kapacitor's TICKScript lambda expression representation and allow the standard set of mathematical relational and logical operators in the boolean expression. For example, following are a few valid boolean conditional expressions for *crit1*, *crit2* and *crit3* in the example template involving two variables:

```
crit1: ("var" == '8') AND ("inst" == '11')
crit2: ("var" == '9') AND ("inst" == '11')
crit3: (var1.value > val1) OR (var2.value <= val2)
```

crit1 filters data points of variable/sensor (say, that measures temperature) with id 8 belonging to instrument with id 11. Similary, *crit2* filters data points of variable/sensor (say, that measures pH value) with id 9 belonging to instrument with id 11. *crit3* is the boolean condition to raise alert which denotes the condition - Temperature greater than val1 OR pH less than equal to val2. The example template can be extended easily for monitoring more variables task/channel. This involves including more crit# variables, joining them and creating a complex lambda expression. For example, to monitor four variables of an instrument in the variable declaration, we would add two more lambda variables $crit4$, $crit5$ in the example template. The variables $crit1$, $crit2$, $crit3$, $crit4$ would represent criteria to filter points for corresponding to four variables' values

Table 2: Streams Channels API

Method	REST API	Description
GET	/v3/channels	Lists Channels
POST	/v3/channels	Creates a Channel
GET	/v3/channels/{id}	Get details of the specified Channel
PUT	/v3/channels/{id}	Update the specified Channel definition
POST	/v3/channels/{id}	Activate/Deactivate the specified Channel
GET	/v3/channels/{id}/alerts	Lists alerts for a given channel
DEL	/v3/channels/{id}	Delete the specified Channel

and $crit5$ would represents the final lambda expression to raise an alert. The lambda expression could be defined as follows:

```
crit5: (var1.value > val1) AND ((var2.value <= val2)
      OR (var3.value > val3)) AND (var4.value > val4)
```

Note that in the *script* field, the actual TICkScript shown in Figure 2 needs to be reformatted to a single line as it is sent in a JSON request body to the Streams API and the Streams service sends it to Kapacitor HTTP template API. We provide these template scripts to users so they can easily write their template tasks.

4.2 Channels

A Channel is an abstraction for pre-processing real-time streaming data to trigger alerts based on a function of the variables and initiate analysis on the data. A channel resource uses a template task and measurement variables from a one or more data streams to define a task with boolean conditions/criteria on the variables' values and a set of actions. A template task can be reused to generate multiple channels with different boolean conditions.

4.2.1 REST API. Table 2 shows Channels REST API to create, update, delete and list channels. Stream resources - projects, sites, instruments, variables, templates - need to be created prior to Channel creation. A Streams service user can create a channel by sending an HTTP POST request to /v3/streams/channels in a JSON formatted request body. The request body comprises of following required fields:

- *channel_id*: a unique channel identifier
- *channel_name*: a channel name
- *template_id*: A pre-created template's identifier which is required to create a channel
- *triggers_with_actions.inst_ids*: a set of instruments ids' for which the channel is created
- *triggers_with_actions.condition*: Boolean conditions on variables' values or measurements for monitoring the variables and raising the alert when conditions are met. The measurements can come from different instruments associated with different projects.
- *triggers_with_actions.action.actor_id*: Abaco's *actor_id*. The actor that performs the analysis will be executed whenever an alarm is triggered.

4.2.2 Channel Design. A channel is an extension to the notion of a task in the Kapacitor. It defines a set of boolean conditions and actions to be initiated when the conditions are met. A boolean condition is defined as a logical expression comprising different variables or functions of variables with thresholds. Once this boolean

expression is evaluated to true, the following set of actions are performed:

- "Alerts" as events are generated, and notifications are sent to the data processing end-point via HTTP POST with details of the data raising the alerts.
- Pre-processing of the incoming alert data.
- Initiating processing of measurements using a Tapis Abaco function, thereby enabling a real-time streaming data event-driven workflow. Such workflows can be used for processing streaming data in real-time, anomaly detection analysis, and time-series machine learning applications.

An important step in channel creation involves parsing the boolean conditional expression and converting it to lambda expression as required in the pre-created template. It also checks if the actor specified in the definition is a valid actor. We will now discuss some key aspects of Channel design.

- **Boolean conditional expression representation in the Channel definition**: For templates, we leveraged Kapacitor Templates API to define a template. The lambda expression in the template used an infix notation. In the channel definition, however, we used a combination of infix-prefix notation to represent the expression. The reason for the notation is the ease of parsing the expression, particularly, key's value, that is, *instrument_id.variable_id* and readability. The format of conditional expression is as follows:

```
<condn>:= {"key": <inst_id.var_id>,
           "operator":<rel_op>,
           "val": threshold_value}
<condn_expr>:=["<logical_op>",
             <condn_expr>|<condn>,
             (optional: <condn_expr>) ]
rel_op := < | <= | > | >= | == | !=
logical_op := AND | OR | NOT
```

An example of a <condn expr>:

```
["AND", {"key":"inst_id1.temp",
         "operator":">", "val": 100},
        ["OR", {"key":"inst_id2.humid",
        "operator":"<", "val": 40},
  {"key":"inst_id2.turb","operator":">",
   "val": 10}
  ]
]
```

- **Boolean expression parser:** A boolean expression parser and converter parses the <condn_expr>, obtains the variable values, and converts it to lambda expression infix notation expression for substituting it to the template. We used a recursive program structure for the parser implementation.
- **Channel Creation**: Kapacitor task API is used to create a task corresponding to a channel definition. For the Kapacitor task definition, it is required to pass JSON formatted TICKScript variables values to be substituted in the template task besides template_id. Streams API converts the channel definition into the Kapacitor task definition using the above parser. When a channel is created, the status is set to "ACTIVE", indicating that the channel is now monitoring the in-coming real-time data.
- **Authorization**: Streams Service has three role levels for controlling channel access as described above.When a user creates a channel, a channel admin role is created in the Security Kernel and the user is granted that role. Admin or managers can grant other roles such as "manager" or "user" based on the channel usage. It is important to note that the user making a create channel request should have atleast a "user" role on the project to which *inst_id* in the HTTP POST JSON request body belongs.

4.3 Alerts

Alerts are the triggered events when a set of boolean conditions are met. For incoming measurement values for different variables or sensors, the channel raises an alert when the values cross certain thresholds. These alerts conditions are defined in the Channel definition and gets substituted in the template. Since Kapacitor performs the boolean conditional expression check, it sends the raised alert information via HTTP POST to the Streams data processing end-point, i.e., /alerts end-point. This end-point, as stated earlier, is defined in the Kapacitor.conf file and used in the template task. Based on the alerts received, Streams pre-process the alerts, and initiates the action as specified in the channel definition. Lists of alerts for a channel can be obtained by making an HTTP GET request to /v3/channels/id/alerts end-point.

4.4 Integrated Architecture

This subsection presents the Tapis V3 Streams API architecture integrated with Kapacitor, Tapis Abaco, and Tapis Meta V3 services to achieve event-driven workflow. Figure 3 shows the integrated architecture and the information flow for alerts generation and initiation of an execution of a pre-registered actor in the Abaco systems. A high-level architecture of Streams API integrated with other components of Tapis V3 and CHORDS is provided in [18]. Before creating a channel and template, the stream's resources (Projects, Sites, Instruments, Variables, and Measurement) need to be defined. To create a template, the user sends an HTTP POST request to the Streams /templates end-point with *template_id* and template task *script*. After template creation, the user sends an HTTP POST request to Streams /channels end-points with *channel_id*, *channel_name*, *template_id* (template created in the previous step), and a set of boolean conditions and set of actions. The template and channel information are stored in the Meta V3 service.

Measurements are posted to Streams /measurements end-point. Streams, in turn, writes the measurements to InfluxDB by sending an HTTP POST request to InfluxDB /write end-point. The values are also copied to the Kapacitor through InfluxDB's Subscription API as they are being written to the database. The Kapacitor task corresponding to the channel, checks the boolean condition. If it is true, then raises an alert and sends the alert to Streams /alerts end-point. The Streams API pre-processes the alert and sends the alert's details in the execution request message to a pre-registered actor in the Abaco service. The Abaco system sends back an *executionId*. On receiving the *executionId*, the Streams API creates an alert and

sends it to the Tapis V3 Meta service to save the data and sends alert information to the user with the *executionId*, which the user can use to track of the execution.

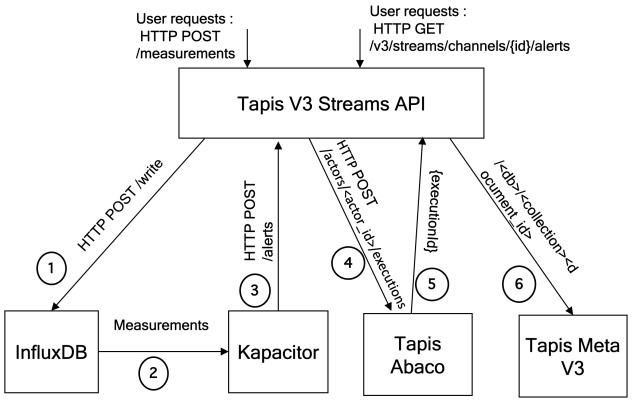

Figure 3: Tapis v3 Streams API's architecture to support event-driven workflow

5 EXPERIMENTAL SETUP AND EVALUATION

This section describes the experimental setup and focuses on the experiments and results to verify and evaluate the new Streams API capability to support the event-driven workflow. It is critical for any real-time monitoring system that if any of its parameters being monitored crosses a predefined threshold, an alert should be generated instantaneously.

5.1 Simulation Setup

We simulated water quality event-driven workflow to verify the correctness and performance of Streams API. The measurements posted to Streams API during simulations were based on our analysis of real-time datasets from two sites: Robert C Byrd Locks and Greenup Locks from the Ohio River (Greenup Pool) Monitoring project. These sites host two instruments (YSI EXO2 Sondes) mounted on the Locks, each with multi-sensor probes monitoring Temperature, Specific Conductance, pH, Turbidity, Dissolved Oxygen, chlorophyll, BGA-Phycocyanin, and Photosynthetic Active Radiation (PAR) to study the quality of water. We identified the measurement range for each variable by processing the datasets with measurements captured and transmitted by the attached data loggers (NexSens 3100-iSIC) every 15 minutes for a period of 30 days as shown in table 3. This approach enabled us to simulate realistic water quality workflow measurements.

To run the simulations, we created an Ohio River Monitoring project with two sites: Robert C Byrd Locks and Greenup Locks, with instruments: Robert C Byrd Sonde and Greenup Sonde, respectively for each site and variables per instrument for measuring Temperature, Specific Conductance, pH, Turbidity, Dissolved Oxygen, Chlorophyll, BGA-Phycocyanin, and Photosynthetic Active Radiation (PAR) leveraging the Streams API python SDK (tapipy [6]) calls. Each of these variables significantly impacts the water quality and the associated habitat dwelling inside it. For example, the water temperature determines the rate of chemical and biological processes. The pH tells us how acidic the water. Dissolved Oxygen affects the survival of plants and animals living in it. Fish and other aquatic animals can perish if they are exposed to water with low Dissolved Oxygen levels (less than 5 milligrams per liter) for an extended time [1]. Similarly, Turbidity is the cloudiness or murkiness in the water. Turbid water is not always harmful, but the sediments and particles in water can be harmful, which needs to be monitored [1].

Table 3: Measurement ranges for Water Quality variables

Variable	Instrument 1	Instrument 2
Battery (V)	8.9 - 15	12.6 - 12.9
Temperature (C)	1.7 - 7.06	2.42 - 7.47
Sp Cond (µS/cm)	195 - 319	247 - 357
PH	7.4 - 8.02	7.29 - 7.93
Turbidity (ntu)	6.9 - 761	11.52 - 264.21
Odo sat. (%)	92 - 112	93.72 - 100.61
Odo (mg/L)	11.28 - 13.6	11.42 - 13.49
Chlorophyll (µg/L)	1.27 - 6.28	1.11 - 6.55
Chlorophyll (RFU)	0.3 - 1.5	0.3 - 2.6
BGA-Phycocyanin (µg/L)	0.43 - 2.55	0.29 - 0.86
BGA-Phycocyanin (RFU)	0.4 - 2.6	0.3 - 0.8
Depth (m)	0.727 - 6.265	0.263 - 2.34
Wiper Pos. (V)	1.18 - 1.21	1.18- 1.23
Cable Power. (V)	8.9 - 14	12.1- 13.7
PAR (µmol/s/m2)	7.4 - 1472	7.4 - 1468

For simulation purposes, we chose to monitor these four variables (Temperature, pH, Dissolved Oxygen, and Turbidity) individually and in combination, satisfying certain logical conditions, by creating active Channels on each of the instruments with Streams API. We used different templates involving one, two, and three variables for creating different channels. Most of the measurements posted to the variables fall in the range specified in Table 3, and few measurements are deliberately posted in a way that they exceed predefined thresholds to generate alerts. We preregistered an Abaco actor, which receives messages from Streams API. The messages received contain one or more measurements that have exceeded the predefined threshold and raised the alert. In case of an alert event, on receiving message from Streams API, the Abaco actor launches the associated container and takes necessary action as programmed in the container image, which could be triggering another job defined in the workflow to run on HPC clusters. Our tests measure the correctness of Streams API supporting event-driven workflow by measuring the number of alerts generated per channel. We calculate the number of alerts generated and match them with the expected numbers to calculate the error rate. The rate at which measurements are posted determines whether Streams services can handle multiple data-points simultaneously and still perform within the expected time and correctness. Although the real experiment posts measurements 15 minutes apart, in our simulations we tried to stress the system, to handle data coming at the rate of less than a second. We will discuss the results from our simulation in the following section.

Table 4: Streams Channels Alerts Design Evaluation

Create Measurement	Num. of Conditions	Inst.1: alerts generated/expected	Inst.2: alerts generated/expected
0.1s	One	13/13	13/13
0.5s	One	12/12	11/11
1s	One	10/10	12/12
0.1s	Two	3/3	5/5
0.5s	Two	2/2	5/5
1s	Two	6/6	1/1
0.1s	Three	6/6	3/3
0.5s	Three	4/4	3/3
1s	Three	3/3	3/3

5.2 Simulation Results

We refer to the instruments, Ohio River Robert C Byrd Locks and Ohio River Greenup Locks as Instrument1 and Instrument2. Measurements are posted to each instrument at the rate of 0.1s, 0.5s and 1s. These instruments are separately monitored with different channels. We categorized the test runs as follows:

5.2.1 Alert generation while monitoring one variable . We created two channels using one-variable template for monitoring both the instruments. Pre-condition set up in both channels was to raise an alert if the temperature value exceeds 9 degrees Celsius. A total of 1400 measurements, 100 values for each variable per instrument, at the rate of 14 values per 0.1s, 0.5s and 1s were posted with the Streams measurements-write API call. At the end of every run, we calculated the number of alerts generated versus the expected number of alerts as shown in Table 4.

5.2.2 Alert generation while monitoring two variables. We created a template with two variables for this run and used it to create two channels listening on both instruments. Two variables chosen for evaluation in this case were temperature and pH. An alert was raised when the temperature exceeds 9 degrees Celsius, and the pH value is less than 5. An AND condition defined in the channel makes sure to raise alert only when both the conditions on the variables evaluate true. Measurements posted in this run were before 1400, 100 values for each variable, with the same sampling rate. Alerts generated versus expected were calculated, and test results for Channel condition, Temp. > 9 AND pH < 5 are recorded in table 4.

5.2.3 Alert generation while monitoring three variables. We created a template with three variables for this run and used it to create two channels listening to their respective instrument. Three variables chosen for evaluation, in this case, were temperature, pH and turbidity. An alert was generated when the condition: Temp. > 9 AND pH < 5 OR Turb. > 500, was satisfied. The number of measurements posted and sampling rate remained the same as in previous cases. The number of alerts generated versus expected was recorded in the test results in Table 4.

6 CONCLUSION AND FUTURE WORK

In this paper we demonstrated that the Streams API's newly developed capabilities (Channels and Templates) successfully support event-driven scientific workflow use cases. We also discussed the Streams service's integration with an alerts automation tool, Kapacitor, and the distributed computing platform, Tapis Abaco. Such integration helps with real-time data monitoring and allows to take necessary actions in case of an event, thereby enabling event-driven workflow. Simulation results obtained from the water quality event-driven workflow verifies the correctness of the new Streams API capabilities. Since the Streams API leverages the CHORDS data model and uses the same InfluxDB's schema, this work shows the CHORDS data service can quickly adopt the alerts' design to enable event-driven workflow.

The Streams API will be available in production supporting real-time event-driven workflows for some of the early adopters: water quality use case stated in Section 3 and automated statewide rainfall mapping for Hawaii [8].

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Office of Advanced CyberInfrastructure, the Tapis Framework:[1931439 and 1931575], Collaborative Research: SS2-SSI: The Agave Platform: An Open Science-As-A-Service Cloud Platform for Reproducible Science NSF ACI #145041.

REFERENCES

[1] 2014. Water measurements. Retrieved Mar 5, 2021 from https://waterwatch.usgs.gov/wqwatch/faq?faq_id=5
[2] 2015. Appalachian Freshwater Initiative (AFI). NSF EPSCoR RII Track 1, 1458952, WV, 2015-2020.
[3] 2016. Sensing and Educating the Nexus to Sustain Ecosystems(SENSE). NSF EPSCoR RII Track 2 FEC, 1632888, WV-KY, 2016-2020.
[4] 2018. Extensible Geospatial Data Framework towards FAIR Science (GeoEDF). NSF CSSI: Framework: Data: HDR, 1835822, 2018-2023.
[5] 2019. *Python Flask Web Framework.* Retrieved Feb 17, 2020 from https://pypi.org/project/Flask/
[6] 2019. tapipy - Tapis V3 Python SDK. Retrieved Mar 8, 2021 from https://github.com/tapis-project/tapipy
[7] 2019. *Tapis.* Retrieved Mar 9, 2021 from https://tapis-project.org/
[8] 2020. Automated Rainfall Mapping in Hawaii. Retrieved May 21, 2021 from https://tapis-project.github.io/tapis-early-adopters-workshop/block2/UH/
[9] 2020. *Influx DB: Time series DB.* https://www.influxdata.com/products/influxdb-cloud/
[10] 2020. *InfluxDB subscription.* Retrieved Mar 8, 2021 from https://docs.influxdata.com/influxdb/v1.8/administration/subscription-management/
[11] 2020. *Kapacitor.* Retrieved Sep 29, 2020 from https://www.influxdata.com/time-series-platform/kapacitor/
[12] 2020. *Ruby on Rails.* Retrieved Feb 17, 2020 from https://rubyonrails.org
[13] 2020. *Tapis Live Docs.* https://tapis-project.github.io/live-docs/ Mar 9, 2021.
[14] 2020. *TICKscript.* Retrieved Sept 29, 2020 from https://docs.influxdata.com/kapacitor/v1.5/tick/
[15] 2021. Apache Spark streaming. Retrieved Mar 8, 2021 from https://spark.apache.org/streaming/
[16] 2021. *Kubernetes: Container Orchestration.* Retrieved Mar 9, 2021 from https://kubernetes.io
[17] 2021. MSQL DB. Retrieved Mar 9, 2021 from https://www.mysql.com
[18] Sean B. Cleveland, Anagha Jamthe, Smruti Padhy, Joe Stubbs, Steven Terry, Julia Looney, Richard Cardone, Michael Packard, Maytal Dahan, and Gwen A. Jacobs. 2020. Tapis v3 Streams API: Time-series and data-driven event support in science gateway infrastructure. *Concurrency and Computation: Practice and Experience* (2020). https://doi.org/10.1002/cpe.6103 First published online 25 Nov 2020.
[19] B. Kerkez et al. 2016. Cloud Hosted Real-time Data Services for the Geosciences (CHORDS). Geoscience Data Journal, 2–4. https://doi.org/doi:10.1002/gdj3.36
[20] J. Stubbs et al. 2018. Rapid Development of Scalable, Distributed Computation with Abaco. Science Gateways Community Institute, 10th International Workshop on Science Gateways.
[21] J. Stubbs et al. 2021. Tapis: An API Platform for Reproducible, Distributed Computational Research. *Future Generation Computer Systems* (2021). accepted.
[22] Theo Zschörnig, Robert Wehlitz, and Bogdan Franczyk. 2020. IoT Analytics Architectures: Challenges, Solution Proposals and Future Research Directions. In *Research Challenges in Information Science*, Fabiano Dalpiaz, Jelena Zdravkovic, and Pericles Loucopoulos (Eds.). Springer International Publishing, Cham, 76–92.