Item 44
git_comments:

1. ToType returns a Go type of the passed in Schema. Types returned by ToType are always of Struct kind. Returns an error if the Schema cannot be converted to a type.
2. Logical Types are for things that have more specialized user representation already, or things like Time or protocol buffers. They would be encoded with the schema encoding.
3. keep the original type for errors. The top level schema for a pointer to struct and the struct is the same.
4. **comment:** TODO(BEAM-9615): implement looking up specific options from the tags.
   label: requirement
5. **comment:** case *pipepb.FieldType_LogicalType: TODO(BEAM-9615): handle LogicalTypes types.
   label: requirement
6. Go field name must be capitalized for export and encoding.
7. parseTag splits a struct field's beam tag into its name and comma-separated options.
8. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to You under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
9. Panics for invalid map keys (slices/iterables)
10. FromType returns a Beam Schema of the passed in type. Returns an error if the type cannot be converted to a Schema.
11. **comment:** case *pipepb.FieldType_IterableType: TODO(BEAM-9615): handle IterableTypes.
    label: requirement
12. Special handling for []byte
13. Package schema contains utility functions for relating Go types and Beam Schemas. Not all Go types can be converted to schemas. This is Go is more expressive than Beam schemas. Just as not all Go types can be serialized, similarly, not all Beam Schemas will have a conversion to Go types, until the correct mechanism exists in the SDK to handle them. While efforts will be made to have conversions be reversable, this will not be possible in all instances. Eg. Go arrays as fields will be converted to Beam Arrays, but a Beam Array type will map by default to a Go slice.
14. must be an atomic type
15. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to You under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

git_commits:

1. **summary:** [BEAM-9615] Add initial Schema to Go conversions.
   message: [BEAM-9615] Add initial Schema to Go conversions.

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
2. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
3. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
   label: code-design
4. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
5. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
6. **summary:** [Go SDK] Beam Schemas

description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas

7. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
8. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
9. **summary:** [Go SDK] Beam Schemas
   description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas
10. **summary:** [Go SDK] Beam Schemas
    description: Schema support is required for advanced cross language features in Beam, and has the opportunity to replace the current default JSON encoding of elements. Some quick notes, though a better fleshed out doc with details will be forthcoming: * All base coders should be implemented, and listed as coder capabilities. I think only stringutf8 is missing presently. * Should support fairly arbitrary user types, seamlessly. That is, users should be able to rely on it "just working" if their type is compatible. * Should support schema metadata tagging. In particular, one breaking shift in the default will be to explicitly fail pipelines if elements have unexported fields, when no other custom coder has been added. This has been a source of errors/dropped data/keys and a simply warning at construction time won't cut it. However, we could provide a manual "use beam schemas, but ignore unexported fields" registration as a work around. Edit: Doc is now at https://s.apache.org/beam-go-schemas

jira_issues_comments:

1. This issue is assigned but has not received an update in 30 days so it has been labeled "stale-assigned". If you are still working on the issue, please give an update and remove the label. If you are no longer

working on the issue, please unassign so someone else may work on it. In 7 days the issue will be automatically unassigned.

2. State of the world slowed down progress on this, but I'm now rolling out PRs for review.
3. **body:** I have end to end working case for schemas of ordinary go user structs. There's still necessary handling around errors, but those are easier to make fluent when we see how users get the code to fail. An issue I didn't quite expect is that there's no way to have top level schemas be pointer types without an extra option or similar. I've worked around it by having a hard option to extract a registered type, but that feels problematic in general. I think I'll introduce a go specific "nillable" option for the top level schema, but the short result is that from external transforms, one essentially always needs to use a value type rather than a pointer type when schemas are involved since there's no good way for the Go SDK to infer that it needs a pointer type at that level of coder, unless the runner doesn't elide the go specific option. Care needs to be taken with the option when decoding row fields, since field types can already be nillable, and we would want to avoid unnecessary pointer to pointers, which would be incorrect in this case.
   label: code-design
4. This issue is assigned but has not received an update in 30 days so it has been labeled "stale-assigned". If you are still working on the issue, please give an update and remove the label. If you are no longer working on the issue, please unassign so someone else may work on it. In 7 days the issue will be automatically unassigned.
5. Been busy with other tasks, but this is still on my plate.
6. I now have Logical types working, and am working on being able to Provision protocol buffer types properly. This wasn't something that was clear from what I read before writing the design doc.
7. This issue is assigned but has not received an update in 30 days so it has been labeled "stale-assigned". If you are still working on the issue, please give an update and remove the label. If you are no longer working on the issue, please unassign so someone else may work on it. In 7 days the issue will be automatically unassigned.
8. Just need to sort out maps and provisioners, and benchmarking them. The tricky bits here is getting the APIs in a good place. And I've been busy with the 2.26.0 release.